Database - Feature #6371

implement SAVE CACHE statement

05/17/2022 09:29 AM - Greg Shah

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Boris Schegolev	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

History

#4 - 05/17/2022 09:41 AM - Greg Shah

The SAVE CACHE language statement allows schema changes to be made to a live database and will read the latest changes and make them available for dynamic query usage. We have a customer that uses SQL DDL and the SQLEngine to make changes to a database. This database is empty by default and is not used except as an extension mechanism for their application's users. The application adds tables and also can add fields to tables they previously added via this same mechanism. The idea is that there is no static 4GL code which references these tables, the tables are for custom extensions per installation which are honored through dynamic queries and some level of configuration/functionality to manage the extension tables.

This is highly related to #3912. In that case, the 4GL code itself is editing the database definition using metadata. But the idea is the same. That application has an installation-specific empty database to which they add tables, fields and indices. Similarly, there is no static 4GL code that uses this extension database, just dynamic queries.

It would make sense to use the same mechanism for both. At least this will include all the configuration updates and DMO generation.

#5 - 05/17/2022 10:02 AM - Greg Shah

To be clear: we are not planning to support the idea of saving this cache into a binary file and sharing it across sessions which is how the 4GL SAVE CACHE works. We intend to just use this statement as a hook to tell us when we need to read the diffs in the schema (recent schema changes) and generate the configuration/DMOs needed so that metadata and dynamic queries will work properly.

#6 - 05/26/2022 11:30 PM - Eric Faulhaber

With SAVE CACHE as our notification hook that something has changed in the database's schema, we would need to respond by invoking a bit of application-specific, custom code. Not sure what that code would look like yet, but its purpose would be to inform FWD of exactly what has changed. This piece is something new; AFAIK SAVE CACHE saves all schema information to file, and doesn't need to discriminate what has changed from what has not. In the absence of this, we would need to compare the new schema state against its previous state and figure out the diffs, which seems like more work (and slower) than it needs to be.

The applications which edit the schema using DDL and JDBC know what aspects of the schema they are changing. This information comes from some form of metadata managed by the applications. We will need a common interface/data structure which that new, custom code will use to package that information for FWD's use.

On the FWD side, we will need to convert new tables into DMO interfaces and update the affected VSTs, so that any metadata queries will provide the most current structural information.

TBD: how to handle:

- modifications of existing schema constructs vs. additions of new constructs?
- table or field deletion?
- index creation, changes or deletion?
- updating FWD's VST state (e.g., _File, _Field, _Index, etc.)?

#7 - 05/27/2022 05:50 AM - Greg Shah

> In the absence of this, we would need to compare the new schema state against its previous state and figure out the diffs, which seems like more work (and slower) than it needs to be.

Sorry, this doesn't make sense to me.

I doubt that speed matters here.

As far as the work to figure out the diffs, it will be less work to do it once using the database than for each customer to write 4GL code to do this. We lose the entire point of this if the customer must submit the changes twice (once via DDL and another time in via the 4GL). Also, the DDL is likely to be written and executed in some other language like Java. To translate this to the 4GL seems a mess. The customer will just want to submit the DDL twice, which means we would then have to parse the DDL. None of this seems to be a clean, long term solution.

Calculating the diffs doesn't seem difficult.

#8 - 05/27/2022 05:35 PM - Eric Faulhaber

Greg Shah wrote:

> > In the absence of this, we would need to compare the new schema state against its previous state and figure out the diffs, which seems like more work (and slower) than it needs to be.
>
> Sorry, this doesn't make sense to me.
>
> I doubt that speed matters here.

Speed always matters. In both cases, this is a runtime feature where a user makes a change and waits for it to be deployed. We need to figure out the change(s) quickly, convert DMOs on the fly, load them, and update VSTs with the changes.

Using JDBC database metadata is the obvious way to implement this. If one knows exactly which schema element(s) to examine, this can work well (assuming each JDBC driver supports this access, which is very likely to be the case for this type of structural data). However, without knowing the exact element(s) which changed, this can be very slow, and the cost is linear to the size of the schema. The JDBC database metadata API is pretty dumb about letting you examine the schema in bulk. It requires many small queries, which really add up as a schema gets large. I know this from experience from querying information for all indices, for example.

The alternative is coming up with a dialect-specific way to collect this metadata quickly from system catalogs, which of course would differ for each database vendor. We potentially still could have a lot of comparison work to do for a large schema, but at least we could craft queries that would

return the schema information in bulk, quickly. This presupposes access to this catalog information is exposed (likely so for most major databases). If exposed, it generally requires a superuser role to access. And it is subject to change between database releases, unless the system tables are officially published and guaranteed not to change. So, this approach is more fragile than the JDBC metadata approach.

Getting some hint as to what changed (a table name or list of names probably would be enough) would go a long way in making either process (but especially the JDBC metadata approach) faster, even if we have to discern the actual changes through before/after diffs.

#9 - 05/28/2022 11:09 AM - Greg Shah

This is mostly a development-time activity. Both customer use cases that we know about are less about the runtime nature and more about someone customizing the system to meet some business need. It is not something done every day or by every user. It is done in preparation for a deployment or possibly after deployment by expert users that are working to extend the system to cover new processes/track additional data. If it is slow, so what? Do we really want to waste time optimizing this case with the cost being a more complex code solution for the customer?

If it takes 5 minutes to process this on a big database that is probably fine. In both known cases, the databases themselves are normally empty and are only filled by these extension tables. We are not likely to see big databases processed this way in the near future, which makes it less likely to matter.

Later on, if this ever becomes a performance problem we can address it then. For now, please plan the simple approach where the customer's 4GL code does not need to submit anything other than the SAVE CACHE notification itself. Generic JDBC metadata seems fine as a solution. I don't see the value in expanding the dialects for this purpose.

#10 - 07/11/2022 11:50 AM - Eric Faulhaber

- Assignee set to Boris Schegolev

This feature is only for use with specially designated databases, whose schemas are mutable. These are special purpose databases which support user defined tables, indices, etc. for purposes such as customer-specific extension information, caching, etc. As such, they generally are smaller databases with relatively small schemas. Primary application databases will not support this feature.

The basic flow of this proposed solution:

1. The converted application invokes some JDBC code external to the application which executes DDL to add, delete, or change a table, field (i.e., column), or index.
   - Multiple such changes are possible in one change set.
   - This external code is a customer's responsibility to create/maintain, though we will need to provide samples to illustrate certain aspects, such as how to modify extent fields. Currently, we have 2 (soon to be 3) options for this:
     - extent fields normalized into a secondary table;
     - extent fields denormalized into multiple columns in the primary table;
     - (coming soon, see #6418) extent fields represented as dialect-specific array column types.
   - We will need to pick one of these (TBD) to use as the standard for these types of mutable databases.
2. The application calls the converted form of the SAVE CACHE 4GL statement. Instead of doing exactly what the 4GL environment does, we use this invocation simply as a hook to notify the FWD runtime that the aforementioned schema change has taken place.
3. Upon receiving the SAVE CACHE notification, the FWD runtime:
   1. Scans the schema and determines what changes have been made.
      - This presupposes we have a baseline representation of the schema before the changes with which to compare the current schema. The form of this is TBD. We do have FWD's implementation of the 4GL metadata tables (e.g., _File, _Field, _Index, etc.), but we may want something more suited to the comparison task.
      - The initial idea (as discussed above) is to use JDBC database metadata for this purpose. The schemas we are contemplating for this task should be relatively small (compared to the applications' main databases' schemas).
   2. Reflects these changes in the DMO interfaces available to the application.
      - New DMO interfaces are easy enough to convert and load at runtime. We do this already for dynamically defined temp-tables. The front-end to this runtime conversion will need to be different, but the back-end, runtime conversion and classloading should be the same or very similar.
      - Changes to existing DMO interfaces pose more of a challenge. These already will be loaded by a classloader. In order to change them while keeping the same interface name, the current interface by that name will need to be unloaded from the JVM, and its replacement re-loaded into the JVM. In Java 8, I think this means one classloader per DMO interface (needs confirmation; I haven't tried this). I

don't know whether there is some feature in Java 11 that makes class unloading easier.
    - Dropping a DMO interface poses similar challenges w.r.t. class unloading, though it is less of an issue, since we can assume the application will not reference dropped tables. Dynamically defined temp-tables today remain loaded as DMO interfaces; they are not explicitly unloaded. However, this is probably ok, because it is likely that they will be defined again and re-used, as other users of the application hit the same code paths which need those temp-tables.
3. Updates FWD's metadata to reflect the schema change. For example, the add/drop/change of a column definition will require an associated change to the meta_field (representing the original _Field) table.

#11 - 07/11/2022 12:06 PM - Boris Schegolev

As I was writing, Eric added the above specs, answering most of my questions :)

A few points from my side:

1. The JDBC metadata approach should be fine. Even if there will be several queries sent to the server, that's no big impact.
2. Also, there's not much difference between loading information about particular objects and loading information about all objects. Considering the databases are expected to be mostly empty, if it's easier to implement bulk loading, we should do that.
3. Dialects should not be used here. We should rely on vendor-independent JDBC representation.

Now for the questions:

1. I understand we should be tracking changes on tables, columns and indexes. Is there anything else we should track?
2. ~~This operation will happen in runtime of the converted application? Whenever SAVE CACHE command is issued, we are expected to trigger the comparison, right?~~ **YES**
3. ~~We will trigger some callback, so we will probably provide some API to subscribe to changes.~~ **NO**, we will be reloading classes, not calling some code.
4. Can you point me to the "dynamically defined temp-tables" support in our code? I'd like to see the implementation details.

#12 - 07/11/2022 12:54 PM - Eric Faulhaber

Boris Schegolev wrote:

> As I was writing, Eric added the above specs, answering most of my questions :)
>
> A few points from my side:
>
> 1. The JDBC metadata approach should be fine. Even if there will be several queries sent to the server, that's no big impact.
> 2. Also, there's not much difference between loading information about particular objects and loading information about all objects. Considering the databases are expected to be mostly empty, if it's easier to implement bulk loading, we should do that.
> 3. Dialects should not be used here. We should rely on vendor-independent JDBC representation.

All good :)

> Now for the questions:
>
> 1. I understand we should be tracking changes on tables, columns and indexes. Is there anything else we should track?

These for certain. Maybe sequences, too, but I'll need to confirm.

1. ~~This operation will happen in runtime of the converted application? Whenever SAVE CACHE command is issued, we are expected to trigger the comparison, right?~~ **YES**
2. ~~We will trigger some callback, so we will probably provide some API to subscribe to changes.~~ **NO**, we will be reloading classes, not calling some code.
3. Can you point me to the "dynamically defined temp-tables" support in our code? I'd like to see the implementation details.

Are you sure? ;)

It starts with an application doing something that requires a dynamic temp-table. This can be an explicit invocation of the CREATE TEMP-TABLE statement, or it can be a host of other things, any of which gets you a handle to a new, primordial temp-table. This handle is then used to flesh out the temp-table definition.

However it gets started, in FWD, you eventually instantiate a TempTableBuilder object (you can follow the c'tor call hierarchy back to the various places which get you here, but the implementation of how the temp-table is defined and prepared for use is downstream from this point. Once you have a TempTableBuilder, converted 4GL code makes a TempTableBuilder.createLike and/or some number of TempTableBuilder.add* calls to define the temp-table. Each of these create/add methods is converted from a specific 4GL counterpart method, which is invoked on the temp-table handle you got earlier.

Once the temp-table is defined, it must be prepared before it can be used. In the 4GL, this is done with the TEMP-TABLE-PREPARE method. In FWD, this translates to one of the TTB.tempTablePrepare method variants. These all eventually delegate to TTB.tempTablePrepareImpl, which in turn invokes DynamicTablesHelper.createDynamicDMO. This is where the runtime conversion takes place.

You will see multiple calls to ConversionPool.runTask. These invoke several TRPL conversion rule-sets. These are the same rule-sets that run during static schema conversion, but they are invoked in a special runtime mode which allows information sharing with the FWD runtime. The last step in this conversion process (see the rule-set defined for the ConversionProfile.BREW_DMO_ADM enum element) is a runtime-only step which uses ASM (https://asm.ow2.io/) to assemble a bytecode array which represents a DMO interface for the dynamically defined temp-table. This bytecode is then loaded by our AsmClassLoader. A companion DMO implementation class is assembled and loaded when we call DmoMetadataManager.registerDmo(dmoIface).

From that point, the DMO interface and implementation classes are loaded in the JVM and available for application code to use them to load data into them and execute dynamically defined queries against them, just like statically defined temp-tables.

Dynamic temp-table definitions are cached, so that if we come upon one for which we've already converted/assembled/loaded the DMO interface and implementation class, we skip that step and just return the existing classes.

I've left out some detail, but hopefully this is enough to get you started. Feel free to ask questions as necessary.

#13 - 07/13/2022 03:14 PM - Eric Faulhaber

- Status changed from New to WIP

Boris Schegolev wrote (in email):

> My only question at the moment is how to simulate the desired state. I guess I need an application that uses SAVE CACHE at some point.

The most basic test case would be a one line 4GL program:

```
save cache {current|complete} <database_name> to <some_path>.
```

For our purposes, everything except the save cache and <database_name> would be ignored by conversion, but it would look something like this in legacy code.

As a very basic starting test, you could mimic the JDBC DDL changes by making some manual changes via a simple database client (e.g., psql for PostgreSQL), then run this converted program to kick off the notification response described in #6371-10. A less manual test could use Direct Java Access from this 4GL test case to invoke a Java method which performs the schema changes via JDBC, before calling save cache ....

This testing approach presupposes several things:

- Conversion and minimal runtime support for the SAVE CACHE statement. That still needs to be designed and implemented. I can help with bootstrapping this. The initial runtime support would be a stub method, that you would more fully implement further.
- A mechanism to make the detection of schema change as simple/efficient as possible. Currently, schema structural state is available at runtime in several forms:
  - An implementation of legacy metadata tables in an embedded H2 database (e.g., meta_file, meta_field, meta_index, etc.), which exposes schema information to converted 4GL business logic. Legacy 4GL code can query these tables, but it is not really optimized for our purpose here. However, this metadata will need to be updated to reflect schema changes detected by this feature.
  - Java annotations in the DMO interfaces themselves, which describe legacy table, field, and index information. This information generally is slower to access directly, so we load it into DmoMeta instances for faster runtime access.
  - Somewhat overlapping facilities like TableMapper, RecordMeta, and DmoMeta, which describe legacy schema information for various runtime purposes. DmoMeta probably has all the metaschema information we need to know about. This may provide a good starting point for this feature, though it's probably not well-structured for the detection/comparison process itself. Currently, this information is considered immutable, though it will have to be updated to reflect the schema changes detected.

#14 - 07/14/2022 05:12 PM - Boris Schegolev

- File 6371.diff added

#15 - 07/14/2022 07:23 PM - Eric Faulhaber

Code review 6371.diff:

The changes seem fine; looks like general code cleanup. Assuming they compile (I didn't apply and confirm), please commit to 3821c.

#16 - 07/18/2022 09:50 AM - Constantin Asofiei

There is a compile error in 3821c/14083:

```
     [exec] [ant:javac] ~/branches/3821c/src/com/goldencode/p2j/persist/DynamicTablesHelper.java:180: error: c
annot infer type arguments for ContextLocal<T>
     [exec] [ant:javac]    private final ContextLocal<Context> context = new ContextLocal<>()
     [exec] [ant:javac]                                                          ^
     [exec] [ant:javac]   reason: cannot use '<>' with anonymous inner classes
     [exec] [ant:javac]   where T is a type-variable:
     [exec] [ant:javac]     T extends Object declared in class ContextLocal
```

Boris: when committing something to a branch, please also post at the appropriate task the revision number.

#17 - 07/18/2022 02:30 PM - Boris Schegolev

Constantin Asofiei wrote:

> There is a compile error in 3821c/14083:
> [...]
>
> Boris: when committing something to a branch, please also post at the appropriate task the revision number.

Fixed in revision 3821c/14085. Used bad compiler version. Sorry for inconvenience.

#18 - 07/27/2022 07:21 AM - Boris Schegolev

- File boris.p added

#19 - 08/01/2022 03:42 PM - Boris Schegolev

- File SchemaComparator.java added

I am attaching a proposal for Schema Comparator service. It would be invoked like:

```
// Once the application starts - this gets information on current DB schema state
SchemaComparator.saveInitialState(database);

// When SAVE CACHE happens, we could do
SchemaComparator comparator = SchemaComparator.getInstanceExistingState(database);
ComparisonResult result = comparator.compareWithPrevious();
if (!result.isSame()) ...
```

Let me know if that fits the common use case.

#20 - 08/03/2022 12:41 AM - Eric Faulhaber

Boris Schegolev wrote:

> I am attaching a proposal for Schema Comparator service. It would be invoked like:
>
> [...]
>
> Let me know if that fits the common use case.

I think so. This looks like a reasonable starting framework. We'll have to work out the details of what the saved state object looks like. That object (or its data in some form) will have to be persistent across FWD server restarts.

Application/server startup is a special case. If the schema is changed while the FWD server is down (which is possible or even likely), we will need to detect the diffs when the application next starts, without the nudge of a SAVE CACHE notification. We have server startup hooks we can leverage for this. A baseline version of the comparable state will need to be created once, at server startup, if it does not already exist.

#21 - 08/03/2022 04:39 PM - Boris Schegolev

Saved state will consist of tables, columns and index lists. This object can be serialized using common methods (YAML, JSON, XML or other format of choice) and stored on the filesystem or as a blob in database. This information would be loaded (or created if it does not exist) on startup.

Please, share any details on startup hooks.

#22 - 08/05/2022 02:35 PM - Boris Schegolev

- File SchemaComparator.java added

Implemented DatabaseState object, Difference object and loading / comparison for tables. Please, review.

#23 - 08/09/2022 10:47 AM - Eric Faulhaber

Boris, please commit all your updates to 6371a. You are the only one changing this branch, so it is safe. This will make it easier for me to review and we can always make changes there. Thanks.

#24 - 08/09/2022 10:56 AM - Eric Faulhaber

Boris Schegolev wrote:

> Saved state will consist of tables, columns and index lists. This object can be serialized using common methods (YAML, JSON, XML or other format of choice) and stored on the filesystem or as a blob in database. This information would be loaded (or created if it does not exist) on startup.

I don't think we have any use of YAML in the project currently, so unless there's a strong technical reason to introduce this, we probably should go with something already in use. I don't have a strong preference between JSON or XML, though I tend to think JSON would be better, as it's typically more compact than XML.

> Please, share any details on startup hooks.

See Server and Context Hooks. This would be a server hook.

#25 - 08/09/2022 11:02 AM - Eric Faulhaber

Eric Faulhaber wrote:

> Boris, please commit all your updates to 6371a. You are the only one changing this branch, so it is safe. This will make it easier for me to review and we can always make changes there. Thanks.

Please review and apply our coding standards. Also, please add javadoc to all members/methods (even private); this will make it easier for a reader to quickly understand the code.

#26 - 08/09/2022 11:20 AM - Tijs Wickardt

Eric Faulhaber wrote:

> Boris Schegolev wrote:
>
> > Saved state will consist of tables, columns and index lists. This object can be serialized using common methods (YAML, JSON, XML or other format of choice) and stored on the filesystem or as a blob in database. This information would be loaded (or created if it does not exist) on startup.
>
> I don't think we have any use of YAML in the project currently, so unless there's a strong technical reason to introduce this, we probably should go with something already in use. I don't have a strong preference between JSON or XML, though I tend to think JSON would be better, as it's typically more compact than XML.

My 2 cents: YAML is a superset of JSON. All JSON can be read by a YAML parser. YAML is for humans, so a "technical reason" isn't present, unless technically required by a FWD customer. The choice between XML and JSON is normally: robustness versus speed. XML has various subformats, like XSD , which is great for datasets like these. You can supply a schema .xsd file for runtime validating and self-documenting the data structure. XML also allows for strong queries, using XPath (v1 - the world, or v2 - Java only). As well as XSLT. JSON is more free/loose, but you can use json-schema or similar for validation. Parsing JSON in the browser, and various generic runtimes, is very much optimized for performance.
Of course you know all this.
Personally I think XML+XSD seems like a good choice here (even if I like JSON for it's ease of use, metadata is something that should be accompanied by a schema, and XSD is more standard than json-schema).

#27 - 08/09/2022 04:59 PM - Boris Schegolev

I pushed revision 6371a/14095 with code to support comparison for tables and columns. Also, extended the javadoc on the SchemaComparator class (will do others later on).

As for the storage, I'm ok with both XML and JSON. Do we want to have a vote on that?

#28 - 08/09/2022 05:45 PM - Greg Shah

XML is more consistent with the rest of the configuration in FWD.

#29 - 08/10/2022 11:19 AM - Eric Faulhaber

Let's go with XML for persisting each "snapshot" state, which will be used for later comparisons. TBD: do we need to preserve more than one generation of these, or do we simply overwrite the last one after we've successfully processed a SAVE CACHE notification?

I've also been reviewing the SchemaComparator code and considering what we need to do downstream from the detection of diffs (see #6371-10,

bullet points 3.2 (DMO interface/class changes) and 3.3 (metadata changes). Each time we compare the current schema state to the previous state (if any), we have to flow the changes through to the DMOs and metadata. For this, we need to consider what inputs we need, as these will need to be additional outputs from the SAVE CACHE process. More on this in follow-up posts...

The "difference" state (currently represented by SchemaComparator$Difference) probably only needs to be in memory. We will need this to drive the downstream changes. Initially, I was thinking we might need to persist it, in case the full SAVE CACHE workflow (i.e. detect differences -> update DMOs and metadata) fails, but I think instead we just start over at the next opportunity.

TBD: how do we determine when the next opportunity is to recover from a failure, since we will not get another SAVE CACHE notification naturally until the next one is invoked explicitly from business logic? Do we raise an error condition which the application is set to handle and leave it to the application to re-invoke SAVE CACHE?

#30 - 08/10/2022 12:35 PM - Eric Faulhaber

After detecting schema differences, we need to inspect/apply/walk these to generate the correct DMO interfaces. DMO interfaces are Java interfaces (see Data Model Objects) which describe the methods used by converted business logic to read and write data to an associated table in the database.

DMO interfaces are generated either during static conversion or (in the case of dynamic temp-tables) at runtime, composed by converted 4GL business logic (see Dynamic Database (Runtime Conversion)). Static conversion of persistent tables and temp-tables produces a Java source file for a DMO interface. Dynamically defined temp-tables do not; their bytecode is assembled on-the-fly and loaded into the JVM with a custom classloader.

The SAVE CACHE process will follow the dynamic temp-table pattern for new and modified tables. That is, there will be no source code generated for DMO interfaces. They will be assembled and loaded into the JVM on-the-fly. Deleted and modified DMO interfaces must first be unloaded. Any 4GL application which uses this technique already will have invented some means of persisting the metadata used to drive their side of the process (i.e., user definition of tables and indices). How we handle the DMOs and ORM is an implementation detail, so there is no dependency or requirement to have Java source code for the DMO interfaces.

While the infrastructure to generate DMO interfaces for dynamic temp-tables already exists, the front end of the process is a bit different than it will be for the DMO interfaces generated in response to a SAVE CACHE notification. The temp-tables are defined by converted business logic calling into various TempTableBuilder (TTB) APIs to define the temp-table. Then TTB.tempTablePrepare is invoked to kick off the on-the-fly conversion process to take the state inside a TTB instance and generate the DMO interface from it (again, see Dynamic Database (Runtime Conversion)) for a high-level description of this flow.

Early on, the dynamic temp-table conversion process uses the TTB instance to generate an in-memory AST which represents the table AST which would come out of the rules/schema/fixups.xml rule-set. This is roughly the equivalent of the temp-table AST which would be preserved as an XML file with a .schema extension, when running static conversion. It is important to note that the TTB contains state that was specified in legacy, 4GL terms, since it was defined by converted 4GL business logic.

From there, we use ConversionPool to run the TRPL rule-sets:

- schema/p2o - creates first pass P2O AST from schema AST. The P2O maps the resources from the schema AST (specified in legacy terms, as it was created from the legacy state stored in the TTB) to Java and SQL terms.
- schema/p2o_post - completes creation of P2O AST
- schema/java_dmo - walks P2O AST to generate Java AST representing the DMO interface
- schema/brew_dmo_asm - walks Java AST to assemble bytecode for DMO interface classfile

Once we have the assembled classfile bytecode, we load it into the JVM using AsmClassLoader. Currently, dynamic temp-tables are only additive; DMO interfaces are never unloaded from the JVM. This is an important difference to consider for this task.

Finally, we register the DMO interface with the MetadataManager, which causes a companion DMO implementation class to be assembled and loaded. We will have to take unloading this into account as well.

We will follow a similar pattern here, except the initial, schema AST will not be generated from a TTB instance. We will need to generate it from the schema information we have gleaned through JDBC instead. In fact, we may want to skip this legacy, schema AST altogether, and generate the P2O AST directly, since the JDBC metadata inspection already will result in information specified in SQL terms, not legacy terms. We still will have to generate the Java annotations in the DMO interface, but I think we can assume the legacy names are essentially the same as the SQL names we find in the JDBC metadata. We will have to generate the Java names (for the DMO class and property names) from those as well.

#31 - 08/10/2022 12:47 PM - Greg Shah

Another difference from dynamic temp-tables: the generated DMOs must survive server restarts (or must be recreated on every server start, which doesn't seem like a great idea).

The reasoning: the SAVE CACHE is only guaranteed to be called once after custom DDL is executed. The server may be restarted hundreds or thousands of times after that and before the next SAVE CACHE. The tables must be usable from 4GL code any any time, not just while the server is still running after the SAVE CACHE statement.

The easiest idea IMO is to persist the generated byte code as a set of .class files which are re-loaded at server startup.

#32 - 08/10/2022 12:49 PM - Greg Shah

Also, I wonder if we should even do this persisting for generated DMOs for dynamic temp-tables (at least if they are heavily used). That is really about performance and is not part of this task.

#33 - 08/10/2022 01:11 PM - Eric Faulhaber

The persistence of the DMOs is a good point. We would want these to load like any other DMOs, if no schema changes had been made offline. Putting aside the dynamic temp-tables for now, we would only want to do a server startup load of DMOs generated by this SAVE CACHE process if the associated tables were not modified or dropped in the meantime.

We will be doing a comparison at server startup, which would catch any schema changes done offline, for which we could not have gotten a SAVE CACHE notification. This probably is not how the current process works for the two applications we initially are targeting, because AFAIK, the table changes are made by users in an environment that would have to have the server running. But we don't know what the future holds, and it is simple enough to treat server startup as an analog to a SAVE CACHE notification.

OTOH, we may want to keep it simple: load all the previously defined DMOs at server startup, and just let the normal processing unload the ones that have changed offline?

#34 - 08/10/2022 01:14 PM - Greg Shah

If the cost is not high, making an implicit SAVE CACHE call at server startup will make the system more resilient/safer.

#35 - 08/10/2022 05:10 PM - Boris Schegolev

Pushed revision 6371a/14096 with JavaDoc and indexes.

#36 - 08/26/2022 03:17 AM - Eric Faulhaber

- File save-cache1.p added

6371a/14098 contains a first pass implementation of conversion support for SAVE CACHE. The statement converts to a static invocation of DatabaseManager.schemaChanged(String ldbName). The destination path/file of the target cache file in the SAVE CACHE statement are ignored, since we are just using this as a notification that a database schema with the given logical database name has changed.

The attached sample 4GL program converts to the following:

```
...

public class SaveCache1
{
   @LegacySignature(type = Type.VARIABLE, name = "ldb")
   character ldb = UndoableFactory.character("p2j_test");
```

```
   /**
    * External procedure (converted to Java from the 4GL source code
    * in save-cache1.p).
    */
   @LegacySignature(type = Type.MAIN, name = "save-cache1.p")
   public void execute()
   {
      externalProcedure(SaveCache1.this, new Block((Body) () ->
      {
         //  variable form
         DatabaseManager.schemaChanged((ldb).toStringMessage());

         //  literal form
         DatabaseManager.schemaChanged("p2j_test");
      }));
   }
}
```

I stubbed out DatabaseManager.schemaChanged and made some related changes. This method is where we need to wire up to the comparator code and further downstream code as described in #6371-10.

#37 - 08/26/2022 12:02 PM - Boris Schegolev

So, the integration part should be something like that, right?

```
   /**
    * Report any changes in database structure compared to last known state.
    *
    * 4GL SAVE CACHE operation invokes this method.
    *
    * @param databaseName
    */
   public static void schemaChanged(String databaseName)
   {
      Database db = new Database(databaseName);
      Comparator comparator = Comparator.getInstanceExistingState(db);
      try {
         ComparisonResult result = comparator.compareWithPrevious();
         if (!result.isSame())
         {
            // TODO
         }
      } catch (IOException e) {
         // TODO
      }
   }
```

#38 - 08/26/2022 12:12 PM - Eric Faulhaber

Yes, something like that, but please note I've already stubbed out DatabaseManager.schemaChanged in 6371a/14098. Please see that. I would expect your code starting with:

```
Comparator comparator = Comparator.getInstanceExistingState(database);
...
```

to slot in where I wrote:

```
// TODO: wire up to the schema comparison and downstream processing
```

If you haven't already, please review my changes to 6371a with:

```
bzr diff -c14098 --using meld
```

#39 - 08/30/2022 03:33 PM - Boris Schegolev

I pushed the latest version (see revision 6371a/14100). I tried to use existing DynamicTablesHelper implementation instead of duplicating the code.

If I understand the code correctly, DynamicTablesHelper.createDynamicDMO() is all that is needed to load a new DMO class during runtime. It seems to be very straightforward - please let me know if I'm missing something.

#40 - 09/05/2022 04:21 PM - Eric Faulhaber

It is not ideal to use DynamicTablesHelper (in its current form) to create persistent table DMO interfaces/classes. I think we need to refactor this class to use much of its functionality as possible, but it needs to be generalized to work with both temp-tables and persistent tables.

Also, we don't want to re-use the temp-table specific front end of the dynamic table definition process (i.e., TempTableBuilder). The TempTableBuilder class is meant to provide an API to converted 4GL code, which that code calls to define a dynamic, 4GL temp-table. We don't need an application-facing API to do this; we already will have collected all such information while inspecting a database schema for changes.

Don't we already have the structure information collected in instances of the Table and Index classes (from the persist.orm.schema.element package)? Would those not be useful inputs into a refactored DynamicTablesHelper class, which we could use to bypass TempTableBuilder?

We need a P2O AST to use as input to the schema conversion process in DynamicTablesHelper. What is the fastest path to get from the information we collect in SchemaComparator to a P2O AST which defines the table and its related index information?

#41 - 09/12/2022 04:05 PM - Boris Schegolev

I updated DatabaseManager to handle registration of the new interface. I am still working on removal of the last call to DynamicTablesHelper - the generateSchemaAst(), which handles for individual fields. Is my understanding correct, that after this effort is complete the whole feature should work? Or is any crucial part still missing?

Code is in revision 6371a/14101.

#42 - 09/13/2022 01:38 AM - Eric Faulhaber

Boris Schegolev wrote:

> I updated DatabaseManager to handle registration of the new interface.

I didn't really mean for DatabaseManager to be the permanent location for the full implementation of this feature; it was more meant to be a launching point for possibly a dedicated class. But for the first draft, let's leave it in DatabaseManager and then we can see how it can be cleaned-up/refactored.

> I am still working on removal of the last call to DynamicTablesHelper - the generateSchemaAst(), which handles for individual fields.

Yes, this is the tricky part. For this task, this call has to be replaced with something that does a similar job, as this is the first input to the dynamic DMO creation. We need to generate either the schema AST (as is done for temp-tables), or ultimately (as a future optimization) the P2O AST, from the information we have collected about the schema.

As you know, when we dynamically convert temp-tables, we use the information fed to the TempTableBuilder APIs from converted business logic. All this information is in "legacy" form: 4GL table, field, and index names. We use this to build up a schema AST, then run TRPL conversion logic against it to create the next evolution of this information: the P2O "peer" AST. This is the code in DynamicTablesHelper.createDynamicDMO which does this:

```
            ...
            Aast root = instance.generateSchemaAst(builder,
                                                   ifaceName,
                                                   sqlTableName,
                                                   legacyTableName,
                                                   clsFields4GLtoORM);

            try
            {
               ConversionPool.Results results;

               results = ConversionPool.runTask(ConversionProfile.P2O, root);
               root = (Aast) results.getStoredObject("p2o.prog");
               ConversionPool.runTask(ConversionProfile.P2O_POST, results, root);

               root = (Aast) results.getStoredObject("p2o.peer");
               root.putAnnotation("permanent", false);
               root.putAnnotation("package", DMO_BASE_PACKAGE);
               ...
```

The call to generateSchemaAst above builds up into AST form the schema information which defines the legacy temp-table. This AST is the equivalent of the schema AST which would have been generated after parsing a DEFINE TEMP-TABLE statement in 4GL code, and then running the rules/schema/fixups.xml TRPL rule-set. The difference with a dynamically defined temp-table is that we do not start with a DEFINE TEMP-TABLE statement, but instead start with a temp-table that is defined in memory (in TempTableBuilder) by calls from converted 4GL business logic to the TempTableBuilder API.

The subsequent calls to ConversionPool.runTask, using the conversion profiles P2O and P2O_POST take that schema AST as input and produce a "peer" P2O AST which contains the same information, plus additional information needed on the Java and SQL side, such as converted table/class, column/property, and index names which are legal for use with Java/SQL. The P2O AST is organized slightly differently, in a form suitable to walk and create a Java DMO interface.

As a possible performance optimization in this dynamic temp-table generation code, I'd eventually like to skip the step which generates the legacy schema AST, and go right to the generation of the P2O peer AST. But that is not for this task, and we probably want to avoid it for the first pass at this task as well, because a lot happens in the P2O and P2O_POST steps, and I don't want to recreate the wheel.

What I am not sure about is from where we gather the legacy information (i.e., the information that goes into that legacy schema AST, and which we ultimately have to expose via the DMO interface annotations -- see the Java class and method annotations of any converted DMO interface to see what I mean). During the static conversion process, this information comes from the exported schema definition (the *.df file). In dynamic conversion, it comes from the business logic. It is not yet clear to me how we get it in this task. Any converted 4GL code running against these tables will be using the legacy names of tables and fields, so we need to know at runtime how this maps to the converted equivalents. We may just need to assume that those names are the same as the ones we find using our JDBC scan of the RDBMS.

> Is my understanding correct, that after this effort is complete the whole feature should work? Or is any crucial part still missing?

Not quite. We still need:

- A solution for unloading obsolete DMO interfaces and implementation classes. This isn't desirable for dynamic temp-tables, because we want those classes available when they are needed across user contexts or when identically structured temp-tables are defined dynamically. But for persistent, shared tables, we can't have multiple versions of the DMO interfaces and classes loaded simultaneously, as this surely will cause problems.
- We also have to update the legacy metadata which is exposed to converted business logic via an embedded H2 database for at least the _File, _Field, _Index, and _Index-field metadata tables. Think of this as an information schema of sorts. 4GL code (converted) can perform queries on these tables to gather metadata about the schema, so it needs to be kept in sync with the changes to the RDBMS. See the MetadataManager class. Note that this is the last priority, because I think we can get significant testing done with the rest of the implementation while we add this support.

#43 - 09/13/2022 01:27 PM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> What I am not sure about is from where we gather the legacy information (i.e., the information that goes into that legacy schema AST, and which we ultimately have to expose via the DMO interface annotations -- see the Java class and method annotations of any converted DMO interface to see what I mean). During the static conversion process, this information comes from the exported schema definition (the *.df file). In dynamic conversion, it comes from the business logic. It is not yet clear to me how we get it in this task. Any converted 4GL code running against these tables will be using the legacy names of tables and fields, so we need to know at runtime how this maps to the converted equivalents. We may just need to assume that those names are the same as the ones we find using our JDBC scan of the RDBMS.

The legacy annotation data is filled in the AST tree by the DynamicTablesHelper as direct node annotation. Look for the calls to Aast.putAnnotation() in the methods generateSchemaAst() (and createDynamicDMO() for a couple) of DynamicTablesHelper. As you will see, the information is extracted from the field map maintained by the TempTableBuilder.

#44 - 09/13/2022 01:49 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> Eric Faulhaber wrote:
>
> > What I am not sure about is from where we gather the legacy information (i.e., the information that goes into that legacy schema AST, and which we ultimately have to expose via the DMO interface annotations -- see the Java class and method annotations of any converted DMO interface to see what I mean). During the static conversion process, this information comes from the exported schema definition (the *.df file). In dynamic conversion, it comes from the business logic. It is not yet clear to me how we get it in this task. Any converted 4GL code running against these tables will be using the legacy names of tables and fields, so we need to know at runtime how this maps to the converted equivalents. We may just need to assume that those names are the same as the ones we find using our JDBC scan of the RDBMS.
>
> The legacy annotation data is filled in the AST tree by the DynamicTablesHelper as direct node annotation. Look for the calls to Aast.putAnnotation() in the methods generateSchemaAst() (and createDynamicDMO() for a couple) of DynamicTablesHelper. As you will see, the information is extracted from the field map maintained by the TempTableBuilder.

Yes, this is where it comes from in the dynamic query case, but we are not using TempTableBuilder as a source of legacy schema metadata in this task. We are scanning the backend database using JDBC to determine when a table/index/column has been added, removed, or modified. This only gives us information about the SQL names of things, not the legacy names.

I spoke with a customer about this today, and they confirmed that the SQL names they are using match the legacy names that are used by the dynamic queries they form in the 4GL business logic for use with these custom tables. So, we must assume that the SQL names in use are the authoritative record of the legacy names as well.

#45 - 09/15/2022 04:22 PM - Boris Schegolev

In the latest version (6371a/14102) I removed dependency on DynamicTablesHelper completely. The code compiles, but is not really tested.

In the current state, we should be able to load information on the schema change and invoke DatabaseManager.schemaChanged(). This is set to process all changed tables and generate new DMOs.

Limitations in the current code (code that I developed, there are probably many parts of surrounding code that I can not validate without deeper debugging):

- Persisting the state we pulled from DB needs finishing. This can be done easily, so, I didn't focus on that.
- Types of columns/fields need more robust translation. This part was incomplete even in the original implementation in DynamicTablesHelper, so I just put a few options to show the intended use. Ideally, there should be a proper and universal translator. It should also support all types coming from various DB types - I suspect running the same code on PostgreSQL and MySQL will return different types.
- Columns/fields need extra information like comments, case sensitivity, etc. This may be vendor-dependent, too, so needs testing.

- Indexes are not handled in DatabaseManager.generateSchemaAst(). Again, this can be added easily once correct behavior of columns/fields is confirmed.
- Unloading obsolete DMO interfaces needs to be implemented. This still needs investigation.
- Error handling and refactoring would be useful, but not critical. For example, we talked about moving internals of DatabaseManager.schemaChanged() to a different class.

Eric, if you are able to pick up any of these tasks, it would be great (or review those areas and provide some direction/tips). Also, you talked about P2O ASTs and legacy schema, which I don't really understand. Any input here is appreciated.

#46 - 09/15/2022 04:55 PM - Ovidiu Maxiniuc

If (better said 'when') we will be able to cache the DMO generated classes we will surely have a boost for the startup of the server when the cache is validated and loaded. IMHO, we should do this by default. More than that, I think we could do the same for the temp-tables: we just need to create a key of the each used temp-table (I think it does not really matter whether they are static or dynamic) and, before doing the standard processing, we should check the map. In case of a hit, load the class with all information needed into memory. Although for permanent and static temporary table the boost would be minimal (the in-memory assembler is fast enough), for dynamic temp-table (where the DMO interface must be first generated before assembling the DMO class) the speed-up should be really visible to the naked eye.

Then, what if we do another step ahead in this direction? We do cache the dynamic converted queries and reuse them in subsequent executions of same lifetime of the server. But what if we use the storage instead do allow already processed dynamic queries to be loaded instead of reconverting them each time? The same mapping we use for in-memory cache could be used for mapping the necessary data on disk. When a dynamic query is requested to be converted, we should check the in-memory cache, on fail, the disk-cache, and only then do the conversion. More than that, when a new server is installed (or updated) we can do a quick set of user connections/requested using latest code to heat-up the disk cache.

The big problem for both these caches (DMOs and dynamic queries) is the validation of data, based on actual fed data AND the version of FWD. The caches must be invalidated when we decide to change the generation/conversion so that the new code will not conflict old binary content. Probably this optimization should be deferred to another task.

#47 - 09/16/2022 02:26 AM - Eric Faulhaber

Boris Schegolev wrote:

> In the latest version (6371a/14102) I removed dependency on DynamicTablesHelper completely. The code compiles, but is not really tested.

I did confirm it compiles. Parts of the implementation are missing and stubbed with TODOs. For instance, getting the baseline/previous state is stubbed, so I can see why this can't be run/tested in its current form. I see the code to fetch the JDBC metadata and to perform the comparison, but I don't see where we get the baseline/previous state with which we are comparing.

> In the current state, we should be able to load information on the schema change and invoke DatabaseManager.schemaChanged(). This is set to process all changed tables and generate new DMOs.

This is the part I intend to work through, to test, and to fill in any missing implementation. But I need that critical first step of being able to pull up a baseline from the last "scan" we did (which might have been an initial, bootstrap scan). I would like you to focus on implementing that missing step with highest priority, so that I have a baseline database state which is necessary to get the full work flow going.

> Limitations in the current code (code that I developed, there are probably many parts of surrounding code that I can not validate without deeper

debugging):

- Persisting the state we pulled from DB needs finishing. This can be done easily, so, I didn't focus on that.

OK, I guess by this you mean what I am talking about above: doing the initial, "bootstrap" scan and storing it for retrieval later, or storing the latest state after scanning the JDBC metadata for a comparison. Don't worry about how we invoke the bootstrap scan, I can take care of that part. I just need something meaningful returned by Comparator.getInstanceExistingState(database), so that the comparison can do its thing.

BTW, I prefer to go back to the name SchemaComparator. I know that is redundant with the package name, and we can avoid any conflict using an import statement, but Comparator is such a well-known and widely-used interface from java.util, I don't want to cause confusion by reusing it in this way.

- Types of columns/fields need more robust translation. This part was incomplete even in the original implementation in DynamicTablesHelper, so I just put a few options to show the intended use. Ideally, there should be a proper and universal translator. It should also support all types coming from various DB types - I suspect running the same code on PostgreSQL and MySQL will return different types.
- Columns/fields need extra information like comments, case sensitivity, etc. This may be vendor-dependent, too, so needs testing.

We will not have enough detail coming back from JDBC metadata to make some of these decisions. For instance, the data type in use may not give us enough information to determine case-sensitivity. In PostgreSQL, we currently use the text database type for both case-sensitive and for case-insensitive string columns. In queries and indices, we wrap these fields in an upper function if case-insensitivity is needed, but this information is not intrinsic to the data type. Short of changing the data type, I'm not sure how we deal with this.

- Indexes are not handled in DatabaseManager.generateSchemaAst(). Again, this can be added easily once correct behavior of columns/fields is confirmed.
- Unloading obsolete DMO interfaces needs to be implemented. This still needs investigation.

After implementing getInstanceExistingState, I would like you to focus on the investigation of how we unload classes from a running JVM as your third highest priority (more on the second priority below). In my mind, this is the biggest unknown left about this feature, and I want to have a strategy for it. The dependencies are tricky. We cannot implement the dropped or modified table use cases without this.

- Error handling and refactoring would be useful, but not critical. For example, we talked about moving internals of DatabaseManager.schemaChanged() to a different class.

I may make some changes like this over the weekend, if there's time.

I would add to your list of missing features that we do not update the metadata we expose to the converted application (e.g., for the legacy _File, _Field, etc. tables) with the changes we've identified through JDBC. This is lower priority, however.

Eric, if you are able to pick up any of these tasks, it would be great (or review those areas and provide some direction/tips).

I intend to work through the flow from start to finish for the use case of an entirely new table, from the point of getting a baseline/previous state, doing the comparison, through to generating and loading the new DMO interface and implementation class for that table. This flow has to be working in the next few days.

As your second highest priority, please go through all of the classes you've added to persist.orm.schema and implement all TODOs which have to do with core functionality (if there are any TODOs left after working through your first priority). Error handling TODOs can wait, I'm talking about implementing missing functionality in your new classes which would block the new table use case I am trying to make work. Don't worry about the code you've adapted from DynamicTableHelpersHelper or DatabaseManager; I'll deal with that. I am less familiar with your new code, so I want you to focus your efforts there.

We can't modify an existing table until we have a strategy to unload the old DMO interface/class from the JVM. However, it essentially will work the same way after the unload, because once the old, modified DMO is gone, the new DMO will be completely generated, as if it were a newly defined DMO. So, once we have the core use case of an entirely new table working, implementing the modify use case will be incremental.

The dropped table use case is likewise a subset of the modified table use case: it is just the unloading of the DMO interface/class which represents the deleted table, without needing to generate/load anything new.

Creation, deletion, and changes to indices essentially all trigger the modified table use case or the new table use case.

Also, you talked about P2O ASTs and legacy schema, which I don't really understand. Any input here is appreciated.

I'm not sure what you are asking, specifically. When I say "legacy" schema, this is how the table/index is defined for the 4GL application. The names are legal to use with the Progress environment, but not necessarily in the post-conversion environment. For instance, Progress table and field names can have an embedded hyphen, which is not legal in SQL. There is a lot of other legacy metadata which is not available from JDBC metadata: things that are used for UI purposes, like a format phrase for the field, validation expressions and messages, column labels, etc.

Since our only source of information is JDBC metadata, we have to make some simplifying assumptions about these types of non-database, legacy metadata. For instance, we might assume that the table and column names we pull from JDBC metadata are identical to the legacy names which will be referenced in 4GL dynamic queries, and that the column labels are the same as the column names, that there are no validation expressions/messages, etc.

During schema conversion, there is a point where this legacy information is gathered in a "schema" AST. This represents the original, legacy table definition, in tree form. At a certain step in the conversion, we generate a "peer" AST, which represents the post-conversion table and DMO interface, in tree form. This is the input to DDL generation (which we don't need here), and DMO generation.

In this feature, the starting input of our conversion is that legacy, schema AST. This is what we are building up in the generateSchemaAst method. We feed that to the runtime conversion pipeline, implemented by ConversionPool, which uses its runTask method to run a subset of the TRPL rules to create the peer P2O AST, then the DMO interface.

If you have specific questions, let me know, but I intend to ensure those parts are working.

#48 - 09/20/2022 03:50 PM - Boris Schegolev

I got rid of TODOs in the code and followed some of suggestions above (naming). The new code now only has a TODO for types handling and another one for class unloading. The latest version is 6371a/14104.

Concerning types you are right, we will have to do some simplifications as loading information from the database is different then using metadata directly. Currently, there are some types implemented so that the flow is clear in the code. I will do some testing and will try to add more types to cover.

I am looking at class unloading right now.

#49 - 09/22/2022 03:16 PM - Boris Schegolev

Class unloading is done (unless I missed something). Unloading the class from JVM itself is only done by the Garbage Collector. As I understand, this is not really needed, we only need to unregister the interfaces from our own class collections (in DatabaseManager, DmoMetadataManager, etc).

Anyway, the code is in 6371a/14105, please review.

#50 - 09/22/2022 03:26 PM - Eric Faulhaber

Boris Schegolev wrote:

> Class unloading is done (unless I missed something). Unloading the class from JVM itself is only done by the Garbage Collector. As I understand, this is not really needed, we only need to unregister the interfaces from our own class collections (in DatabaseManager, DmoMetadataManager, etc).

That doesn't sound right. AFAIK, the ClassLoader instance which loaded the class maintains a reference (in a Vector, IIRC) to every class it has loaded, and the garbage collector can't unload a class while there is any hard reference to it. The last time I looked into this, the conclusion I came to was that the ClassLoader itself had to be garbage collected, but that was a while ago.

> Anyway, the code is in 6371a/14105, please review.

Will do, thanks.

#51 - 09/22/2022 03:31 PM - Boris Schegolev

Eric Faulhaber wrote:

> the ClassLoader itself had to be garbage collected

That is correct. We can re-instantiate the class loader, so that the old instance goes away and can be garbage-collected. But first we need to get rid of all references in our code, which I hope the latest change does. Please, let me know if it's not so.

#52 - 09/23/2022 01:34 PM - Boris Schegolev

I implemented class unloading through delegates as was suggested in TODO in AsmClassLoader.

The code is in 6371a/14106.

#53 - 09/28/2022 02:06 AM - Eric Faulhaber

I reverted rev 14106 locally because it breaks the singleton contract of AsmClassLoader and thus breaks server startup. I'm focusing first on the core functionality and we can come back to the class unloading.

Regarding the serialization and deserialization of the database state, I have a few notes:

- I am curious what went wrong with using XML as the storage mechanism.
- I don't think we want to use a thread-local Jackson ObjectMapper with a JSON file stored in a shared location. I think we need to do some synchronization of the DMO generation work, so that we don't have multiple contexts defining/loading/redefining/reloading DMOs at the same time. This needs to be managed cleanly.

But again, I'm putting that off as secondary for now to get the basics working.

We need a way to bootstrap when there is no initial/previous state available. Currently, we immediately fail with a FileNotFoundException when the JSON file can't be found. How are we meant to compose the initial, bootstrap database state? Do we call SchemaComparator.compare(null)?

#54 - 09/28/2022 05:53 PM - Eric Faulhaber

FYI, I am making major changes to SchemaComparator. Please do not modify this class until I check it in.

#55 - 09/29/2022 02:28 AM - Eric Faulhaber

- File save-cache2.p added

I've made changes to SchemaComparator, mostly formatting to match our coding standards, and some functional to get further along. I've committed this to 6371a/14107. It does not yet get through the initial collection of information, however.

How are you testing? Are you able to convert and run simple 4GL test cases in FWD? I started testing using the save-cache1.p program attached to this task earlier, but the database which that program references (p2j_test) is a mess on my system after testing many other programs. So, I created an even simpler version (attached), which references a new, empty database named mutable.

It gets to the point of getting index info via JDBC, but the invocation of DatabaseMeta.getIndexInfo is incorrect, so we hit an NPE:

```
java.lang.NullPointerException
        at org.postgresql.core.Utils.escapeLiteral(Utils.java:71)
        at org.postgresql.jdbc.PgConnection.escapeString(PgConnection.java:969)
        at org.postgresql.jdbc.PgDatabaseMetaData.escapeQuotes(PgDatabaseMetaData.java:1018)
        at org.postgresql.jdbc.PgDatabaseMetaData.getIndexInfo(PgDatabaseMetaData.java:2394)
        at com.mchange.v2.c3p0.impl.NewProxyDatabaseMetaData.getIndexInfo(NewProxyDatabaseMetaData.java:3352)
        at com.goldencode.p2j.persist.orm.schema.SchemaComparator.getAndCompareIndexes(SchemaComparator.java:4
19)
        at com.goldencode.p2j.persist.orm.schema.SchemaComparator.compare(SchemaComparator.java:188)
        at com.goldencode.p2j.persist.orm.schema.SchemaComparator.compareWithPrevious(SchemaComparator.java:16
4)
        at com.goldencode.p2j.persist.DatabaseManager.schemaChanged(DatabaseManager.java:849)
        at com.goldencode.testcases.SaveCache2.lambda$execute$0(SaveCache2.java:25)
        at com.goldencode.p2j.util.Block.body(Block.java:636)
        at com.goldencode.p2j.util.BlockManager.processBody(BlockManager.java:8582)
        at com.goldencode.p2j.util.BlockManager.topLevelBlock(BlockManager.java:8250)
        at com.goldencode.p2j.util.BlockManager.externalProcedure(BlockManager.java:529)
        at com.goldencode.p2j.util.BlockManager.externalProcedure(BlockManager.java:500)
        at com.goldencode.testcases.SaveCache2.execute(SaveCache2.java:23)
        at com.goldencode.testcases.SaveCache2MethodAccess.invoke(Unknown Source)
        at com.goldencode.p2j.util.ControlFlowOps$InternalEntryCaller.invokeImpl(ControlFlowOps.java:8454)
        at com.goldencode.p2j.util.ControlFlowOps$InternalEntryCaller.invoke(ControlFlowOps.java:8425)
        at com.goldencode.p2j.util.ControlFlowOps.invokeExternalProcedure(ControlFlowOps.java:5905)
        at com.goldencode.p2j.util.ControlFlowOps.invokeExternalProcedure(ControlFlowOps.java:5764)
        at com.goldencode.p2j.util.ControlFlowOps.invoke(ControlFlowOps.java:943)
        at com.goldencode.p2j.main.StandardServer$LegacyInvoker.execute(StandardServer.java:2270)
        at com.goldencode.p2j.main.StandardServer.invoke(StandardServer.java:1727)
        at com.goldencode.p2j.main.StandardServer.standardEntry(StandardServer.java:611)
        at com.goldencode.p2j.main.StandardServerMethodAccess.invoke(Unknown Source)
        at com.goldencode.p2j.util.MethodInvoker.invoke(MethodInvoker.java:156)
        at com.goldencode.p2j.net.Dispatcher.processInbound(Dispatcher.java:783)
        at com.goldencode.p2j.net.Conversation.block(Conversation.java:422)
        at com.goldencode.p2j.net.Conversation.run(Conversation.java:232)
        at java.lang.Thread.run(Thread.java:750)
```

The problem is that we are trying to get all index info for all tables, but that API doesn't work that way. We need to specify a particular table. So, instead of doing this:

```
      // Load indexes from DB
      ResultSet indexes = conn.getMetaData()
                               .getIndexInfo(null, null, null, false, false);
```

...we need to do something like this:

```
      // Load indexes from DB
      ResultSet indexes = conn.getMetaData()
                               .getIndexInfo(null, null, "my_table", false, false);
```

This is an unfortunate design of this JDBC API, since it forces us to loop through all the known tables and collect their index information one by one.

The setup for testing involves a few changes:

- a mutable namespace must be added to the schema configuration in cfg/p2j.cfg.xml;
- an empty mutable.df file must be added to the project's data directory;
- runtime configuration (i.e., a database section) for the mutable database must be added to directory.xml.

With those changes, the attached save-cache2.p file can be converted and run for testing/debugging.

#56 - 09/29/2022 04:24 AM - Boris Schegolev

Eric Faulhaber wrote:

> The problem is that we are trying to get all index info for all tables, but that API doesn't work that way. We need to specify a particular table.

OK, I'll fix that ASAP.

#57 - 09/29/2022 04:28 AM - Boris Schegolev

BTW, what database are you connected to? MySQL, Postgres, ...?

#58 - 09/29/2022 06:37 AM - Boris Schegolev

I pushed a fix for the NPE as revision 14108.

As for the flow, the intended use goes like this:

1. saveInitialState() is called upon server startup. This overwrites the last known state file. It's a blocking operation. This can be skipped if we are sure that the temporary file representing the current database step already exists.
2. The application waits for schemaChanged() to be invoked by the code converted from SAVE CACHE. If processing is successful, this overwrites the database state file and in-memory last known state.
3. Consecutive schemaChanged() follow previous step.
4. Server restarts should pick up the last known state from the file. Or, it can be reloaded from DB - see step 1.

#59 - 09/29/2022 10:51 AM - Eric Faulhaber

Boris Schegolev wrote:

> BTW, what database are you connected to? MySQL, Postgres, ...?

So far, I've only tested with PostgreSQL. If different databases implement the JDBC metadata APIs differently, we will have to abstract this using the Dialect class hierarchy.

#60 - 09/29/2022 04:27 PM - Boris Schegolev

I pushed several fixes for the comments above:

- SchemaComparator file access synchronization
- ClassLoader delegates are not instances of AsmClassLoader any more
- Initial loading does not fail if database state file does not exist - empty database state is used instead

Code is in revision 14109.

#61 - 09/30/2022 02:16 AM - Eric Faulhaber

Boris Schegolev wrote:

> I pushed several fixes for the comments above:
>
> - SchemaComparator file access synchronization
> - ClassLoader delegates are not instances of AsmClassLoader any more
> - Initial loading does not fail if database state file does not exist - empty database state is used instead
>
> Code is in revision 14109.

Please back out all your AsmClassLoader changes. It is badly broken and prevents the server from starting. I had to revert all changes locally to continue testing.

The initial loading is working now for a simple database, but serialization of the database state does not work. Jackson gets itself into a bad state and throws StackOverflowException eventually. I don't want to spend any time getting this to work, because the intended serialization was to XML, not JSON. Please explain what you tried with XML, and what wasn't working.

#62 - 09/30/2022 06:45 AM - Boris Schegolev

I reverted all Class Loader modifications. We'll have to get back to that later.

Also, I reimplemented and tested serialization, it's now XML. Originally, I had problems with added dependencies for XML support, so using JSON was an easier path. Now, I used SAX/XmlHelper that is already used in the project, but it creates some boilerplate code to maintain. Anyway, it's working now in revision 14110.

#63 - 09/30/2022 10:57 AM - Eric Faulhaber

Thanks for the fixes in 14110. The last known state for my very simple database looks like this persisted:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<tables>
  <table name="book">
    <attribute name="recid" type="int8"/>
    <index columns="recid" name="book_pkey"/>
  </table>
</tables>
```

One question: is attribute always synonymous with a column?

I hit an NPE when trying to register the table, because the first difference added to the collection of tables to reload in DatabaseManager.schemaChanged is null. This seems to be caused by a missing check (namely, && difference.getNewElement() instanceof Table) at DatabaseManager:861.

#64 - 09/30/2022 04:23 PM - Boris Schegolev

Eric Faulhaber wrote:

> One question: is attribute always synonymous with a column?

I introduced constants to make it clear and easily configurable (see version 14111). But yes, I use attributes and columns as synonyms. Usually, I try to avoid generic words like tables and columns, as you end up with conflicting names for example with UI components. Here it's ok to call it whatever you like :)

As for the NPE, I need to do more testing.

#65 - 10/03/2022 03:33 AM - Eric Faulhaber

I have committed revision 14112, which fixes the NPE and refactors the code for this feature out of DatabaseManager and into SchemaCheck. The conversion is updated accordingly, to convert SAVE CACHE to SchemaCheck.run. I am continuing my testing and fixing.

#66 - 10/11/2022 05:00 PM - Boris Schegolev

Pushed revision 14113. Now it's going past generateSchemaAst() call. I can't test further because of some misconfiguration, ConversionPool does not get instantiated.

#67 - 10/18/2022 04:58 PM - Boris Schegolev

I have successfully loaded a table / new DMO into my server runtime. Code is in 6371a/14114 revision.

#68 - 10/21/2022 07:10 PM - Boris Schegolev

I pushed an update that handles the properties of columns (types, nullable, default, etc.), provides null-safe reading of attributes in state.xml, plus minor refactoring (pulled out some constants, etc.) - see revision 6371a/14115.

I still want to do:

1. Generate ASTs for Indexes (marked as TODO in SchemaCheck)
2. Refactor SchemaComparator to conditionally write data into state.xml - so that we don't have to write IFs for every property.
3. Implement class unloading

Also, I wanted to ask what are format and extent properties/annotations of columns/attributes. I didn't find much info on those.

#69 - 10/25/2022 04:21 PM - Boris Schegolev

Pushed point 2 from list above. See revision 6371a/14116.

Point 1 is still in progress.

#70 - 10/28/2022 05:15 PM - Boris Schegolev

I pushed support for indexes (revision 6371a/14117). There's an issue with detecting if the index is primary - this information is not provided by the base API. It can be pulled from another API call, but the result has to be merged. Not sure how important that is for our current goal.

I will be looking into class unloading now.

#71 - 10/28/2022 06:06 PM - Eric Faulhaber

Boris Schegolev wrote:

> I pushed support for indexes (revision 6371a/14117). There's an issue with detecting if the index is primary - this information is not provided by the base API. It can be pulled from another API call, but the result has to be merged. Not sure how important that is for our current goal.

That's a good question. Knowing which index is the primary index can be important when converting dynamic queries at runtime, because the primary index can be selected under certain conditions, when determining how the results of a particular query should sort. However, I'm not sure how we determine which index is considered the primary index, because that concept in Progress is not the same as it is in SQL.

In SQL, the primary index would be the unique index on the primary key. But in Progress, the primary index can be whichever index the schema definer deems to be the primary index. There is no information in a SQL schema that would tell us which index that would be (unless there is only one). Progress must have some rule to pick one, when it accesses tables in a SQL database via its SQL engine. I would guess it is probably the first one it encounters for a given table, however it inspects the schema.

We'll need some input from users of this feature to know which index to pick, if there is more than one.

#72 - 11/03/2022 07:12 PM - Boris Schegolev

I pushed support for class unloading. I still need to do some refactoring and fix a java.io.NotSerializableException, but other than that the server successfully loads and unloads DMOs.

Latest revision is 6371a/14118.

So, are we now feature-complete? Do we cover customer's expectations/usecases with current implementation?

#73 - 11/04/2022 04:58 PM - Boris Schegolev

Serialization Exception was a secondary issue. It was actually caused by unsuccessful class lookup. This is now fixed.

Also, JavaDoc is updated. Revision number is 6371a/14119.

#74 - 11/14/2022 06:37 PM - Boris Schegolev

- % Done changed from 0 to 90

Pushed revision 6371a/14121. Based on testing, added more mappings for column types and moved mapping to Dialect plus minor cleanup.

#75 - 11/17/2022 02:32 PM - Eric Faulhaber

FYI, I am re-working the class unloading, which will not work currently in "real" use scenarios. When we create a DMO proxy to use the new DMO interface/class with a record buffer, the proxy factory caches the DMO interface, preventing the garbage collector from reaping it.

I also don't want to create a class loader instance per loaded class. Only some classes loaded by AsmClassLoader need to be unloadable, and I'd

like them logically grouped together in a class loader instance. There are other users of AsmClassLoader which do not need unloadable classes. I am addressing that at the same time as the proxy issue.

#76 - 11/30/2022 04:07 PM - Boris Schegolev

- Status changed from WIP to Review

- % Done changed from 90 to 100

#77 - 12/01/2022 01:10 AM - Eric Faulhaber

- % Done changed from 100 to 80

- Status changed from Review to WIP

I found that I was unable to convert test cases using 6371a/14119, with an error during annotations:

```
Optional rule set [customer_specific_annotations_prep] not found.
WARNING: Null annotation (package-base) for block [BLOCK] @0:0 (420906795009)
./save-cache3.p
Elapsed job time:  00:00:00.942
ERROR:
com.goldencode.p2j.pattern.TreeWalkException: ERROR!  Active Rule:
-----------------------
      RULE REPORT
-----------------------
Rule Type :   WALK
Source AST:  [ block ] BLOCK/ @0:0 {420906795009}
Copy AST  :  [ block ] BLOCK/ @0:0 {420906795009}
Condition :  qname.startsWith(qprefix)
Loop      :  false
--- END RULE REPORT ---

    at com.goldencode.p2j.pattern.PatternEngine.run(PatternEngine.java:1077)
    at com.goldencode.p2j.convert.TransformDriver.processTrees(TransformDriver.java:574)
    at com.goldencode.p2j.convert.ConversionDriver.back(ConversionDriver.java:540)
    at com.goldencode.p2j.convert.TransformDriver.executeJob(TransformDriver.java:962)
    at com.goldencode.p2j.convert.ConversionDriver.main(ConversionDriver.java:1066)
Caused by: java.lang.NullPointerException
    at com.goldencode.expr.CE5713.execute(Unknown Source)
    at com.goldencode.expr.Expression.execute(Expression.java:373)
    at com.goldencode.p2j.pattern.Rule.apply(Rule.java:500)
    at com.goldencode.p2j.pattern.NamedFunction.execute(NamedFunction.java:453)
    at com.goldencode.p2j.pattern.AstSymbolResolver.execute(AstSymbolResolver.java:795)
    at com.goldencode.p2j.pattern.CommonAstSupport$Library.execLib(CommonAstSupport.java:1851)
    at com.goldencode.expr.CE5711.execute(Unknown Source)
    at com.goldencode.expr.Expression.execute(Expression.java:373)
    at com.goldencode.p2j.pattern.Rule.apply(Rule.java:500)
    at com.goldencode.p2j.pattern.RuleContainer.apply(RuleContainer.java:590)
    at com.goldencode.p2j.pattern.RuleSet.apply(RuleSet.java:98)
    at com.goldencode.p2j.pattern.PatternEngine.apply(PatternEngine.java:1638)
    at com.goldencode.p2j.pattern.PatternEngine.processAst(PatternEngine.java:1554)
    at com.goldencode.p2j.pattern.PatternEngine.processAst(PatternEngine.java:1488)
    at com.goldencode.p2j.pattern.PatternEngine.run(PatternEngine.java:1041)
    ... 4 more
```

Wanting to pick up all the changes which have gone into 3821c since 6371a was created, I rebased 6371a to 3821c/14403. The problem remains. I tracked it down to (post-rebase) revision 6371a/14426. Before this commit, conversion works:

```
Optional rule set [customer_specific_annotations_prep] not found.
./save-cache3.p
## INFO: ./save-cache3.p: assuming DOWN set to 1 for frm1 at KW_FORM
/home/ecf/projects/testcases/uast/src/com/goldencode/testcases/name_map.xml
Elapsed job time:  00:00:02.107
```

Note the absence of the warning:

```
WARNING: Null annotation (package-base) for block [BLOCK] @0:0 (420906795009)
```

The stack trace associated with this error is:

```
java.lang.Throwable
    at com.goldencode.ast.AnnotatedAst.putAnnotationImpl(AnnotatedAst.java:4496)
    at com.goldencode.ast.AnnotatedAst.putAnnotation(AnnotatedAst.java:3063)
    at com.goldencode.ast.XmlFilePlugin.readAnnotations(XmlFilePlugin.java:843)
    at com.goldencode.ast.XmlFilePlugin.createAst(XmlFilePlugin.java:786)
    at com.goldencode.ast.XmlFilePlugin.readAst(XmlFilePlugin.java:683)
    at com.goldencode.ast.XmlFilePlugin.loadTree(XmlFilePlugin.java:353)
    at com.goldencode.ast.XmlFilePlugin.loadTree(XmlFilePlugin.java:451)
    at com.goldencode.ast.AstManager.loadTree(AstManager.java:324)
    at com.goldencode.p2j.pattern.PatternEngine.processAst(PatternEngine.java:1487)
    at com.goldencode.p2j.pattern.PatternEngine.run(PatternEngine.java:1041)
    at com.goldencode.p2j.convert.TransformDriver.processTrees(TransformDriver.java:574)
    at com.goldencode.p2j.convert.ConversionDriver.back(ConversionDriver.java:540)
    at com.goldencode.p2j.convert.TransformDriver.executeJob(TransformDriver.java:962)
    at com.goldencode.p2j.convert.ConversionDriver.main(ConversionDriver.java:1066)
```

It looks to be occurring when the test case's AST is loaded from XML for the Code Conversion Annotations phase. The revision has this entry in the repo:

```
revno: 14426
author: Boris Schegolev <bs@goldencode.com>
committer: Eric Faulhaber <ecf@goldencode.com>
branch nick: 6371a
timestamp: Tue 2022-10-25 22:13:19 +0200
message:
  #6371 implement SAVE CACHE statement

   - refactoring in XmlHelper
   - used XmlHelper in SchemaComparator
   - test for SchemaComparator
   - minor cleanup in Elements
added:
  test/com/goldencode/p2j/persist/
  test/com/goldencode/p2j/persist/orm/
  test/com/goldencode/p2j/persist/orm/schema/
  test/com/goldencode/p2j/persist/orm/schema/SchemaComparatorTest.java
modified:
  src/com/goldencode/p2j/persist/orm/schema/SchemaComparator.java
  src/com/goldencode/p2j/persist/orm/schema/element/Attribute.java
  src/com/goldencode/p2j/persist/orm/schema/element/Element.java
  src/com/goldencode/p2j/persist/orm/schema/element/Index.java
  src/com/goldencode/util/XmlHelper.java
```

Given the nature of the error and the fact that, of the list of modified files, only XmlHelper is used during static conversion, I suspect the changes to XmlHelper.java are related to the regression.

#78 - 12/05/2022 12:51 AM - Eric Faulhaber

Eric Faulhaber wrote:

> I found that I was unable to convert test cases using 6371a/14119, with an error during annotations:
>
> [...]

Boris, any headway on this regression? Please post your findings so far.

#79 - 12/05/2022 06:07 AM - Boris Schegolev

My suspicion is the logical change in XmlHelper.setAttribute() - I added a check for value.isEmpty() and it probably doesn't match the expectations for conversions. Those rely on some attributes to be empty and still be written into XML tags.

I made changes so that XmlHelper provides 2 sets of methods (default and *SkipEmpty), but I can't get the complete build - I get Task :ant-native FAILED. I'm still investigating.

#80 - 12/08/2022 01:06 PM - Eric Faulhaber

Revision 6371a/14431 fixes some issues, including the following with the DMO interface assembly:

- a package path issue (the schema name was repeated);
- an interface hierarchy issue (must extend DataModelObject, rather than Temporary).

Along with some other fixes, we can now create a record buffer for the new DMO.

The next issue is that the SchemaDictionary is not properly updated with a new table's information, so the parser cannot recognize a buffer reference in a dynamic query. The infrastructure for this exists for dynamically generated temp-tables, but we need something similar for permanent tables.

I am using the following test case (save-cache3.p):

```
save cache current "mutable" to "/tmp/mutable.csh".

define var h-buf  as handle          no-undo.
define var h-qry  as handle          no-undo.
define var h-fld  as handle          no-undo.
define var qrytxt as char            no-undo format "x(64)" view-as fill-in.
define var i      as int             no-undo.
define var flds   as int             no-undo.
define var d      as char   extent 5 no-undo.

form
   "Query:" qrytxt
   with frame frm1 centered no-labels title "Query Test".

update qrytxt with frame frm1.

create buffer h-buf for table "test".
create query h-qry.

h-qry:set-buffers(h-buf).
h-qry:query-prepare(qrytxt:screen-value).
h-qry:query-open().
h-qry:get-first().

do while h-buf:available:
   flds = h-buf:num-fields.
```

```
   if flds > 5 then flds = 5.
   do i = 1 to flds:
      h-fld = h-buf:buffer-field(i).
      d[i] = string(h-fld:buffer-value).
   end.
   display d[1] d[2] d[3] d[4] d[5]
      with frame frm2 10 down.
   down with frame frm2.
   h-qry:get-next().
end.
```

This presumes the existence of a database named mutable, which contains a table test, with at least a standard recid column. The UI is primitive, but it allows for the display of up to 5 fields of data. After creating the test table and inserting some rows manually, I can then type a query predicate into the fillin (e.g., for each test) and get the results below the fillin.

That's the theory, at least, currently, the program reports the error: test must be a quoted constant or an unabbreviated, unambiguous buffer/field reference for buffers known to query . (7328). It then hangs and has to be killed. This is due to the SchemaDictionary issue noted above. I am continuing to work on this problem.

#81 - 12/08/2022 04:33 PM - Eric Faulhaber

Boris, if you have a fix for the conversion regression, please commit it.

#82 - 12/09/2022 08:16 AM - Boris Schegolev

Regression is fixed, current revision number is 14434.

#83 - 12/21/2022 02:11 AM - Eric Faulhaber

Revision 6371a/14438 works around the SchemaDictionary issue described in #6371-80, but the workaround is limited, and is not meant to be a permanent solution. The revision also begins some rework of the dialect integration: I would like each dialect to implement its own translateToParmType method, which translates only those data types which are used by static conversion (or their direct aliases) for that dialect. Any other data type should translate to null, and the calling code should treat a null return from this method as an error.

I have begun a document describing FWD's implementation of the SAVE CACHE feature, which eventually will become part of the FWD documentation. I need to add information to the document which describes the current limitations of the feature, and the exact rules which must be followed by a database developer to create a database schema which conforms with the conventions FWD expects. We will need to enforce these rules in the execution of the SAVE CACHE statement.

#84 - 12/21/2022 11:30 AM - Boris Schegolev

I pushed a cleanup for PostgreSQL dialect - there were MySQL types mixed in.

#85 - 12/22/2022 04:26 AM - Eric Faulhaber

Boris, please work on the validation of the mutable database requirements described in the SAVE CACHE wiki.

Note that we will need a more flexible implementation of the SQL->4GL data type mappings than we currently have, since, in several cases, the same SQL type can map to multiple 4GL types. For example, in the PostgreSQL dialect, the text SQL type can represent the 4GL types character, clob, or comhandle. character is by far the most common mapping, but the others can be used on occasion. We will need to come up with some sensible convention which allows us to identify which 4GL type is intended for a given SQL type.

Note that the MariaDbDialect is out of date in branch 3821c (and thus in branch 6371a), and MariaDbLenientDialect does not exist in these branches at all. That is because the support for MariaDB was started in 3821c, but now is being maintained in branch 6129b (another sub-branch of 3821c). We have to figure out how/when to get these branches rationalized/merged. For now, please focus only on the PostgreSQL dialect and the validation mechanism.

For the one-to-many SQL->4GL type mappings, consider how we might implement validation rules using sensible conventions to enable us to choose the correct mapping. Most likely, this will be some sort of naming convention requirement. Please document your ideas here. But start by mapping the SQL type to the first 4GL type listed in the table for that dialect.

#86 - 12/22/2022 05:09 PM - Boris Schegolev

I am implementing one-to-many SQL->4GL type mappings. I would only suggest to use column's comment property for 4GL types instead of naming convention. This would work similarly to how query hints work in Oracle DB - you can specify a hint in the comment so that FWD knows which type you prefer to use. If you don't specify it, FWD will pick one by itself. This should work fine for Postgres and MySQL/Maria. Does it make sense like that?

As for branching, in git I would rebase 6371a onto 6129b. Not sure how to do that in Bazaar and don't want to break it (much). :)

#87 - 12/23/2022 01:32 AM - Eric Faulhaber

Boris Schegolev wrote:

> I am implementing one-to-many SQL->4GL type mappings. I would only suggest to use column's comment property for 4GL types instead of naming convention. This would work similarly to how query hints work in Oracle DB - you can specify a hint in the comment so that FWD knows which type you prefer to use. If you don't specify it, FWD will pick one by itself. This should work fine for Postgres and MySQL/Maria. Does it make sense like that?

That seems like a great idea, as long as column-level comments are supported in create table statements in all the databases we support or plan to support in the near/medium term; at minimum:

- PostgreSQL
- MariaDB
- SQL Server
- Oracle
- Db2
- H2

Please confirm this is the case.

If so, I could envision this concept to be used for any 4GL-specific metadata that needs to be conveyed to FWD from the database schema, using a key/value notation in the column-level comment. Let's start with the data type and determine what else we might need after this is implemented. A comment should not be required if the defaults are used, only for less common choices. I'll add this to the wiki.

#88 - 12/23/2022 09:45 AM - Greg Shah

> As for branching, in git I would rebase 6371a onto 6129b. Not sure how to do that in Bazaar and don't want to break it (much). :)

bzr provides support for rebase. However, we need SAVE CACHE to be in 3821c.

#89 - 12/23/2022 11:15 AM - Boris Schegolev

Eric Faulhaber wrote:

> That seems like a great idea, as long as column-level comments are supported in create table statements in all the databases we support or plan to support in the near/medium term; at minimum:
>
> - PostgreSQL
> - MariaDB
> - SQL Server
> - Oracle
> - Db2
> - H2
>
> Please confirm this is the case.

All mentioned databases support column comments/remarks. Also, comments are available through the JDBC API, so unless there are some limitations on individual implementations, we shouldn't have any issues with that. I pushed a basic implementation - see revision 14440.

Greg Shah wrote:

> bzr provides support for rebase. However, we need SAVE CACHE to be in 3821c.

True. Let's focus on Postgres for now and add other dialects when they are merged into 3821c.

#90 - 12/27/2022 05:28 PM - Boris Schegolev

I updated the documentation / WIKI page. A few questions here:

1. What's the TODO for Surrogate Primary Key about?
2. What is the issue with timezoned types (timestamp with time zone)?
3. In dialects, I prefer to throw an exception if an unsupported type is provided to translateToParmType(). Is there any reason to return null instead of throwing an exception right away?

#91 - 12/28/2022 02:09 PM - Eric Faulhaber

Boris Schegolev wrote:

> I updated the documentation / WIKI page. A few questions here:
>
> 1 What's the TODO for Surrogate Primary Key about?

Before you changed it to "TODO: Surrogate Primary Key", the note for 8-byte integer read, "TODO: need a differentiating convention; see also Surrogate Primary Key". So, this was not meant to be a TODO about the surrogate primary key per se, so much as I was pointing to the description of the way we identify the surrogate primary key described in Surrogate Primary Key section of the SAVE CACHE wiki. That is, the surrogate primary key must be identified by those rules and differentiated from a normal int64 or other 4GL data type. The surrogate primary key must be present in every table, but should not be treated as a legacy 4GL field at all. I've updated the notes to link to the Surrogate Primary Key section.

> 2 What is the issue with timezoned types (timestamp with time zone)?

We need to support them ultimately, but they are more complicated, because in the dialects we currently support, they are treated as two columns in the SQL schema (one for the timestamp, and one for the original time zone).

> 3 In dialects, I prefer to throw an exception if an unsupported type is provided to translateToParmType(). Is there any reason to return null instead of throwing an exception right away?

Then throw PersistenceException in the dialect code, and wrap that in an ErrorConditionException in the calling code (e.g. SchemaCheck). The persistence layer (mostly) works this way, throwing PersistenceException or a subclass (checked exceptions) in the lower levels of the persistence code, and wrapping those in one of the ConditionException concrete subclasses (unchecked exceptions) at higher levels. The latter are meant for use in runtime code which is closer to the converted business logic.

#92 - 12/28/2022 04:41 PM - Boris Schegolev

Eric Faulhaber wrote:

> Before you changed it to "TODO: Surrogate Primary Key", the note for 8-byte integer read, "TODO: need a differentiating convention; see also Surrogate Primary Key". So, this was not meant to be a TODO about the surrogate primary key per se, so much as I was pointing to the description of the way we identify the surrogate primary key described in Surrogate Primary Key section of the SAVE CACHE wiki. That is, the surrogate primary key must be identified by those rules and differentiated from a normal int64 or other 4GL data type. The surrogate primary key must be present in every table, but should not be treated as a legacy 4GL field at all. I've updated the notes to link to the Surrogate Primary Key section.

Yes, I didn't understand the Surrogate Primary Key note, so my changes made even more mess :) Thanks for the wiki update!

> 2 What is the issue with timezoned types (timestamp with time zone)?

> We need to support them ultimately, but they are more complicated, because in the dialects we currently support, they are treated as two columns in the SQL schema (one for the timestamp, and one for the original time zone).

Makes sense.

> Then throw PersistenceException in the dialect code, and wrap that in an ErrorConditionException in the calling code (e.g. SchemaCheck). The persistence layer (mostly) works this way, throwing PersistenceException or a subclass (checked exceptions) in the lower levels of the persistence code, and wrapping those in one of the ConditionException concrete subclasses (unchecked exceptions) at higher levels. The latter are meant for use in runtime code which is closer to the converted business logic.

Working on that.

#93 - 12/28/2022 04:47 PM - Boris Schegolev

Also, for the primary key: I guess we need to add a validation before registering a new table to check that the table has a properly defined recid. Do I just add a manual validation (if else throw Exception style) or is there some rule engine that can validate a new table/DMO?

#94 - 12/28/2022 07:57 PM - Greg Shah

> 2 What is the issue with timezoned types (timestamp with time zone)?

> We need to support them ultimately, but they are more complicated, because in the dialects we currently support, they are treated as two columns in the SQL schema (one for the timestamp, and one for the original time zone).

We have 2 customers needing this and I have no confirmation from both that we can defer this implementation.

As a quick first pass, may I suggest that we implement 2 configuration items using this comment approach:

- 4GL field name
- 4GL data type

Define a protocol by which these two items can be defined consistently. I would suggest that we define simple codes for each data type. The DATETIME-TZ will need 2 codes (e.g. DT-TZ1 for the core DATETIME part and DT-TZ2 for the TZ offset).

If there is no comment, we should implement our best efforts default behavior. The 4GL name is the same as the SQL name, the type is inferred. It is possible that some 4GL types can't be represented using the default approach (e.g. DATETIME-TZ) but that is OK.

If there are any other critical configuration values that cannot be determined by defaults (decimal precision and digits?), then we can add those to the list of items.

#95 - 12/29/2022 12:14 AM - Eric Faulhaber

Boris Schegolev wrote:

> Also, for the primary key: I guess we need to add a validation before registering a new table to check that the table has a properly defined recid.

Yes, and not just for the primary key. All the requirements in Mutable Database Requirements section must be met. Anything encountered outside of

those expected requirements is an error.

> Do I just add a manual validation (if else throw Exception style) or is there some rule engine that can validate a new table/DMO?

There is no existing rule engine for this. Each requirement must be tested for manually (if else throw Exception style).

#96 - 12/29/2022 03:23 PM - Boris Schegolev

I pushed an update for error handling and other minor changes - rev. 14441.

Greg Shah wrote:

> We have 2 customers needing this and I have no confirmation from both that we can defer this implementation.
>
> As a quick first pass, may I suggest that we implement 2 configuration items using this comment approach:
>
> - 4GL field name
> - 4GL data type
>
> Define a protocol by which these two items can be defined consistently. I would suggest that we define simple codes for each data type. The DATETIME-TZ will need 2 codes (e.g. DT-TZ1 for the core DATETIME part and DT-TZ2 for the TZ offset).

That can be done, I added a TODO in the code. Do we have an example of the table structure where such 2-field DTTZ storage is used? First column is DATETIME and the other is INT denoting the offset?

> If there are any other critical configuration values that cannot be determined by defaults (decimal precision and digits?), then we can add those to the list of items.

Yes, we are missing DECIMALS for decimal types. Also, FORMAT and EXTENT annotations are not applied. Should they also be provided as metadata?

#97 - 12/30/2022 11:00 AM - Greg Shah

> Do we have an example of the table structure where such 2-field DTTZ storage is used? First column is DATETIME and the other is INT denoting the offset?

Just define a field of DATETIME-TZ in any table to see this. The 2-column SQL representation is always the way the DATETIME-TZ 4GL type is represented.

> Yes, we are missing DECIMALS for decimal types. Also, FORMAT and EXTENT annotations are not applied. Should they also be provided as metadata?

Eric will have to say, but I suspect these are needed. Also, CASE-SENSITIVE may be needed.

#98 - 12/30/2022 06:11 PM - Boris Schegolev

I added an implementation for SchemaValidator + unit test for it. See revision 14442.

#99 - 01/02/2023 05:34 PM - Eric Faulhaber

Boris Schegolev wrote:

> > If there are any other critical configuration values that cannot be determined by defaults (decimal precision and digits?), then we can add those to the list of items.
>
> > Yes, we are missing DECIMALS for decimal types. Also, FORMAT and EXTENT annotations are not applied. Should they also be provided as metadata?

DECIMALS can be gleaned from the data type; e.g., numeric(50,2) (PostgreSQL) or decimals(50,2) (MariaDB). The second parameter represents the scale, which equates to the DECIMALS option in the 4GL.

FORMAT is optional and there are defaults set for each type if it is not explicitly defined (e.g., "x(8)" for a CHARACTER type). We can use metadata (i.e., SQL comments) for this.

Default EXTENT representation is something we're in the midst of changing right now. The current default behavior is to store these fields in a secondary table named <primary_table>__<extent_size>, linked to the primary table by foreign key. So, if the original table foo contains a field bar with extent 5, the secondary table would be named foo__5. It would contain 5 rows per record in the primary table, with 3 columns:

- parent__id (foreign key to the primary table);
- bar (the actual data of each element in the array);
- list__index (the 0-based index of each element in the array).

If there was another extent 5 field defined in the original table (e.g., foo.bat[5]), a bat column would be added to the foo__5 secondary table.

If there was another extent field of a different size (e.g., foo.baz[7]), a separate secondary table for that (and any other extent 7 fields) would exist, named foo__7.

This is what we've named the "normalized" approach, though it isn't exactly normalized. Just about everyone we've asked wants us to get rid of this and replace it with something better, usually for reasons of performance and schema cleanliness.

For database dialects which provide first class support for arrays, we are considering using these, pending testing for performance and usability.

For database dialects which do not provide first class support for arrays, we will default to our "denormalized" approach, which essentially expands an extent field into separate columns in the primary table, one per extent element. There is a default naming convention for these. Taking the foo example given above, the primary table would have the following columns, representing the expanded extent fields:

- bar_1, bar_2, ... bar_5
- bat_1, bat_2, ... bat_5
- baz_1, baz_2, ... baz_7

However, we cannot rely on this naming convention alone to identify expanded extent fields, since there is nothing preventing an unrelated, scalar field from having a name like bar_1 or baz_7. Also, one can define custom names for denormalized extent fields using conversion schema hints, so the names can be any legal SQL name. Thus, we will need to rely on comments for this metadata as well. Please look at how the DMO getter methods for extent fields are annotated in DMO interfaces for statically converted tables. The SQL comments would need to be similar to this.

#100 - 01/02/2023 09:47 PM - Greg Shah

Let's assume that only the denormalized extents will be supported for now. It doesn't make sense to add the more complex normalized support when we are trying to get rid of it.

#101 - 01/03/2023 02:27 PM - Eric Faulhaber

Greg Shah wrote:

> Let's assume that only the denormalized extents will be supported for now. It doesn't make sense to add the more complex normalized support when we are trying to get rid of it.

Agreed. I didn't mean to imply we needed to support the normalized for SAVE CACHE, but I wanted to describe it as one of the forms of extent field support, since it is currently the default (soon to be deprecated, hopefully).

#102 - 01/03/2023 02:35 PM - Eric Faulhaber

The changes to support the DMO interface/class unloading and re-use at the proxy level are turning into something more complex than I want to tackle at this time.

Since the definition of custom tables is primarily an offline or at least a less common activity, I think it is ok for our first pass at this feature to create

new classes, rather than trying to unload existing DMO classes and re-use their names for modified tables. The DMOs are generally pretty small and simple, are finite in number, and are not persisted across JVM runs. So, this approach shouldn't cause a notable memory issue.

Boris, please implement this change. I don't think we need to roll back the AsmClassLoader changes, because they are additive and we will want to support the unloading/re-use eventually. We just won't use the new unloading capability for now, until we have time to make the ProxyFactory catch up.

#103 - 01/04/2023 06:38 PM - Boris Schegolev

I disabled DMO unloading in SchemaCheck, although I don't fully understand reasoning behind it.

Also, I reworked handling of DECIMALS, now the responsibility for decimals annotation is on the Dialect. I tested it, seems to be working fine for mentioned types.

Updated revision number is 14444.

#104 - 01/05/2023 02:25 AM - Eric Faulhaber

Boris Schegolev wrote:

> I disabled DMO unloading in SchemaCheck, although I don't fully understand reasoning behind it.

The idea (in the short term) is to change the way the DMO interface is named, so that a unique interface is created for each new or changed table. The reason is that removing the strong references to the obsolete interface and classes from the AsmClassLoader is not enough to get these classes to be garbage collected. In real use within FWD, there are many other references to these classes. We don't have time to untangle all this right now, so the simpler solution is to make sure the names are always unique, like we do with the dynamically created temp-tables.

In the longer term, we want to clean up all the other strong references to these classes from the runtime, but that is lower priority, because this is an offline or lightly used feature, and a few extra simple classes in memory are not a huge burden. Plus, a server restart will create only the needed DMO interfaces and classes.

So, please implement a unique interface naming scheme, similar to what we have in DynamicTablesHelper.createDynamicDMO. These interface and implementation class names are not exposed to the user; they are only available through a debugger or with debug-level logging.

> Also, I reworked handling of DECIMALS, now the responsibility for decimals annotation is on the Dialect. I tested it, seems to be working fine for mentioned types.

Good.

> Updated revision number is 14444.

A few notes on the dialects...

- Please remove the implementation of Dialect.translateToParmType and throw UnsupportedOperationException instead. The mappings truly vary by dialect, so having a default implementation is just confusing.
  - Once we have all the dialects supported, we will remove the implementation altogether and make the method abstract at the Dialect level.
  - This approach generally holds for all methods in Dialect. Unless there is an implementation of some feature that represents an ANSI standard or is very commonly used across most databases, the implementations should be in the subclasses.
- In P2JPostgreSQLDialect.translateToParmType, please remove any type that does not represent one of the data types used by the conversion. For instance, we never use CHAR or VARCHAR, nor any floating point types, nor any auto-increment types for generated PostgreSQL DDL. There are others in that switch statement that don't look right (e.g., BPCHAR, BLOB; IIRC, we use TEXT and OID, respectively), but perhaps this is the way the JDBC driver reports them? The best way to test this is to:
  - create a 4GL table with a field of every type;
  - export the schema to a p2j_test.df file;
  - temporarily replace the p2j_test.df file in the "old" testcases project;

- convert the original save-cache1.p test case using this DF file;
- create a PostgreSQL table with the generated DDL;
- run the converted save-cache1.p program in the debugger and see which type strings are passed to P2JPostgreSQLDialect.translateToParmType.

Any type strings in the switch statement that are not accounted for with that test need to be eliminated, so that they will cause a validation error.

We will need the same done for the two MariaDB dialects, once we have the branches rationalized.

We are trying to merge 3821c to trunk this week. We need to determine whether it is safe to merge the SAVE CACHE implementation into it. Please make the above changes and we'll see how close we are.

#105 - 01/06/2023 04:46 AM - Eric Faulhaber

I committed 6371/14445, which fixes the raising of an error condition for a schema validation failure. Now save-cache3.p looks like this (at the bottom of the ChUI), in the event of a problem with primary key validation:

```
** Surrogate primary key must be present. Table needs to have a recid column

Procedure complete. Press space bar to continue.
```

BTW, we should include the table name in this error message. Also, the message doesn't convey the actual problem. In this case, the primary key existed, but it was named wrong. This will be confusing to users.

#106 - 01/06/2023 06:56 AM - Boris Schegolev

Eric Faulhaber wrote:

> So, please implement a unique interface naming scheme, similar to what we have in DynamicTablesHelper.createDynamicDMO. These interface and implementation class names are not exposed to the user; they are only available through a debugger or with debug-level logging.

Done in rev 14446.

- Please remove the implementation of Dialect.translateToParmType and throw UnsupportedOperationException instead. The mappings truly vary by dialect, so having a default implementation is just confusing.

Also done in rev 14446.

> There are others in that switch statement that don't look right (e.g., BPCHAR, BLOB; IIRC, we use TEXT and OID, respectively), but perhaps this is the way the JDBC driver reports them?

I am testing that. I know that some types are changed on the way (SERIAL is represented as INT4 + sequence in Postgres). The JDBC driver seems to stick to basic types, so just INT4 in this case.

> Any type strings in the switch statement that are not accounted for with that test need to be eliminated, so that they will cause a validation error.

Yes, I'll do that.

#107 - 01/06/2023 12:30 PM - Eric Faulhaber

Upon a validation failure, the failing table needs to be removed from the last known DatabaseState. Otherwise, if another schema check is run without making schema changes, it will find that nothing has changed, and will assume that the table it has in memory from the last check was ok.

I don't think we should be throwing ErrorConditionException from SchemaValidator.validate. We should be working with checked exceptions up to the point where converted code needs to be notified of the error. So, I would say throw ValidationException from SchemaValidator.validate, catch it in SchemaCheck.registerTable, and raise the error condition there. Something like:

```
try
{
   SchemaValidator.validate(table);
}
catch (ValidationException exc)
{
   // remove the failing table from the last known database state
   // ...

   // report the error
   ErrorManager.recordOrThrowError(exc);

   return;
}
```

The return is necessary, because ErrorManager.recordOrThrowError will not throw an exception if we are in silent mode (i.e., SAVE CACHE was run with the NO-ERROR option), and we don't want the table registration to continue in this case.

#108 - 01/06/2023 04:45 PM - Boris Schegolev

Yes, I met this problem a few times. I will make the change you suggest. It will not be that straightforward because the validation happens "far away" from the last known state - we might have to expose it. Or have a method to remove just one table. I'll think of something.

#109 - 01/09/2023 06:11 PM - Boris Schegolev

In the end the solution is pretty simple - see revision 14448.

#110 - 01/10/2023 04:12 AM - Eric Faulhaber

Revision 14449 fixes the table name and legacy table name when creating the schema AST. Only the DMO interface name should be unique every pass. The table name and legacy table name must be consistent across runs, so that dynamic 4GL queries can use the legacy name, and the SQL generated for queries uses the correct SQL name.

Note that once the surrogate primary key is identified, it should *not* show up as a legacy field. Currently, when we generate the schema AST, we are treating it as any other attribute/column detected in the scan. This is not correct, as this column should not be exposed to 4GL code as a legacy field.

Please make sure save-cache3.p works correctly after each commit. Currently, for a table that is defined in PostgreSQL like this:

```
mutable=# \d test
                Table "public.test"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 recid  | bigint  |           | not null |
 f1     | integer |           |          |
 f2     | text    |           |          |
Indexes:
    "test_pkey" PRIMARY KEY, btree (recid)
    "idx__test_f1" btree (f1)
```

... and containing records like this:

```
mutable=# select * from test order by f1;
 recid | f1  |      f2
-------+-----+--------------
     1 |   1 | one
     2 |  11 | eleven
     3 | 111 | eleventy-one
(3 rows)
```

... we get output from that test program like this:

```
   ┌──────────────────────────────Query Test──────────────────────────────┐
   │Query: for each test                                                  │
   └──────────────────────────────────────────────────────────────────────┘

┌──────────────────────────────────────────────┐
│d[1]     d[2]     d[3]     d[4]     d[5]     │
│──────── ──────── ──────── ──────── ──────── │
│1        one      1                          │
│11       eleven   2                          │
│111      eleventy 3                          │
│                                             │
│                                             │
│                                             │
│                                             │
│                                             │
│                                             │
│                                             │
└─────────────────────────────────────────────┘

Procedure complete. Press space bar to continue.
```

The d[3] column represents the surrogate primary key column and should not be there. It was detected by this code, which dynamically iterates

through all available fields in the test buffer:

```
do while h-buf:available:
   flds = h-buf:num-fields.
   if flds > 5 then flds = 5.
   do i = 1 to flds:
      h-fld = h-buf:buffer-field(i).
      d[i] = string(h-fld:buffer-value).
   end.
   display d[1] d[2] d[3] d[4] d[5]
      with frame frm2 10 down.
   down with frame frm2.
   h-qry:get-next().
end.
```

We need to avoid this by having something like this in SchemaCheck.generateSchemaAst:

```
      ...

      // add field nodes to table
      Map<String, Aast> fieldAsts = new HashMap<>();
      for (Attribute field : table.getAttributes())
      {
         if (field.isPrimaryKey())
         {
            // skip surrogate primary key
            continue;
         }
         ...
```

#111 - 01/10/2023 04:15 AM - Eric Faulhaber

Also, please review the data type checking in the PostgreSQL dialect again, and apply the methodology I described in #6371-104 to determine the valid data types. The FLOAT4 and FLOAT8 types are not right (no floating point types are used by conversion), and I don't think BPCHAR is right, either. I think we use TEXT for CLOBS, so we probably need to differentiate CLOB types through SQL comments.

#112 - 01/11/2023 04:04 AM - Boris Schegolev

Simply skipping surrogate primary key column seems to break index handling, rules are failing during AST validation. I am still investigating the proper way around it.

Eric Faulhaber wrote:

> and I don't think BPCHAR is right, either.

PostgreSQL shows some character type as BPCHAR, so I added it at some point. My DB is pretty messy at the moment, so I won't double check :)

#113 - 01/13/2023 02:48 AM - Eric Faulhaber

Boris, with rev 14450, I get this in an infinite loop when I run save-cache3.p:

```
[01/13/2023 02:36:47 EST] (com.goldencode.p2j.util.TransactionManager:SEVERE) Abnormal end; original error:
java.lang.IllegalArgumentException: Failed to create DMO implementation for (interface com.goldencode.p2j.pers
ist.dynamic.mutable.book_1)
        at com.goldencode.p2j.persist.orm.DmoMetadataManager.registerDmo(DmoMetadataManager.java:246)
        at com.goldencode.p2j.persist.SchemaCheck.registerTable(SchemaCheck.java:341)
        at com.goldencode.p2j.persist.SchemaCheck.processChanges(SchemaCheck.java:220)
        at com.goldencode.p2j.persist.SchemaCheck.run(SchemaCheck.java:140)
        at com.goldencode.p2j.persist.SchemaCheck.run(SchemaCheck.java:118)
        at com.goldencode.testcases.SaveCache3.lambda$execute$0(SaveCache3.java:59)
        at com.goldencode.p2j.util.Block.body(Block.java:636)
        at com.goldencode.p2j.util.BlockManager.processBody(BlockManager.java:8649)
        at com.goldencode.p2j.util.BlockManager.topLevelBlock(BlockManager.java:8315)
        at com.goldencode.p2j.util.BlockManager.externalProcedure(BlockManager.java:543)
        at com.goldencode.p2j.util.BlockManager.externalProcedure(BlockManager.java:516)
        at com.goldencode.testcases.SaveCache3.execute(SaveCache3.java:55)
        at com.goldencode.testcases.SaveCache3MethodAccess.invoke(Unknown Source)
        at com.goldencode.p2j.util.ControlFlowOps$InternalEntryCaller.invokeImpl(ControlFlowOps.java:9104)
        at com.goldencode.p2j.util.ControlFlowOps$InternalEntryCaller.invoke(ControlFlowOps.java:9060)
        at com.goldencode.p2j.util.ControlFlowOps.invokeExternalProcedure(ControlFlowOps.java:6142)
        at com.goldencode.p2j.util.ControlFlowOps.invokeExternalProcedure(ControlFlowOps.java:6000)
        at com.goldencode.p2j.util.ControlFlowOps.invoke(ControlFlowOps.java:1258)
        at com.goldencode.p2j.util.ControlFlowOps.invoke(ControlFlowOps.java:865)
        at com.goldencode.p2j.main.StandardServer$LegacyInvoker.execute(StandardServer.java:2255)
        at com.goldencode.p2j.main.StandardServer.invoke(StandardServer.java:1714)
        at com.goldencode.p2j.main.StandardServer.standardEntry(StandardServer.java:619)
        at com.goldencode.p2j.main.StandardServerMethodAccess.invoke(Unknown Source)
        at com.goldencode.p2j.util.MethodInvoker.invoke(MethodInvoker.java:156)
        at com.goldencode.p2j.net.Dispatcher.processInbound(Dispatcher.java:784)
        at com.goldencode.p2j.net.Conversation.block(Conversation.java:422)
        at com.goldencode.p2j.net.Conversation.run(Conversation.java:232)
        at java.lang.Thread.run(Thread.java:750)
Caused by: java.lang.ArrayIndexOutOfBoundsException: 0
        at com.goldencode.p2j.persist.orm.RecordMeta.readPropertyMeta(RecordMeta.java:547)
        at com.goldencode.p2j.persist.orm.RecordMeta.<init>(RecordMeta.java:241)
        at com.goldencode.p2j.persist.orm.DmoClass.assembleImplementation(DmoClass.java:442)
        at com.goldencode.p2j.persist.orm.DmoMetadataManager.registerDmo(DmoMetadataManager.java:242)
        at com.goldencode.p2j.persist.SchemaCheck.registerTable(SchemaCheck.java:341)
        at com.goldencode.p2j.persist.SchemaCheck.processChanges(SchemaCheck.java:220)
        at com.goldencode.p2j.persist.SchemaCheck.run(SchemaCheck.java:140)
        at com.goldencode.p2j.persist.SchemaCheck.run(SchemaCheck.java:118)
        at com.goldencode.testcases.SaveCache3.lambda$execute$0(SaveCache3.java:59)
        at com.goldencode.p2j.util.Block.body(Block.java:636)
        at com.goldencode.p2j.util.BlockManager.processBody(BlockManager.java:8649)
        at com.goldencode.p2j.util.BlockManager.topLevelBlock(BlockManager.java:8315)
        at com.goldencode.p2j.util.BlockManager.externalProcedure(BlockManager.java:543)
        at com.goldencode.p2j.util.BlockManager.externalProcedure(BlockManager.java:516)
        at com.goldencode.testcases.SaveCache3.execute(SaveCache3.java:55)
```

```
        at com.goldencode.testcases.SaveCache3MethodAccess.invoke(Unknown Source)
        at com.goldencode.p2j.util.ControlFlowOps$InternalEntryCaller.invokeImpl(ControlFlowOps.java:9104)
        at com.goldencode.p2j.util.ControlFlowOps$InternalEntryCaller.invoke(ControlFlowOps.java:9060)
        at com.goldencode.p2j.util.ControlFlowOps.invokeExternalProcedure(ControlFlowOps.java:6142)
        at com.goldencode.p2j.util.ControlFlowOps.invokeExternalProcedure(ControlFlowOps.java:6000)
        at com.goldencode.p2j.util.ControlFlowOps.invoke(ControlFlowOps.java:1258)
        at com.goldencode.p2j.util.ControlFlowOps.invoke(ControlFlowOps.java:865)
        at com.goldencode.p2j.main.StandardServer$LegacyInvoker.execute(StandardServer.java:2255)
        at com.goldencode.p2j.main.StandardServer.invoke(StandardServer.java:1714)
        at com.goldencode.p2j.main.StandardServer.standardEntry(StandardServer.java:619)
        at com.goldencode.p2j.main.StandardServerMethodAccess.invoke(Unknown Source)
        at com.goldencode.p2j.util.MethodInvoker.invoke(MethodInvoker.java:156)
        at com.goldencode.p2j.net.Dispatcher.processInbound(Dispatcher.java:784)
        at com.goldencode.p2j.net.Conversation.block(Conversation.java:422)
        at com.goldencode.p2j.net.Conversation.run(Conversation.java:232)
        at java.lang.Thread.run(Thread.java:750)
```

This must be what you were seeing when you reported save-cache3.p was not exiting normally. In fact, it was never getting past the SAVE CACHE statement, causing an abnormal end, and then retrying. You need to tail the server log when you run test cases. Some change in rev 14450 seems to have broken the runtime schema conversion.

#114 - 01/13/2023 03:53 AM - Eric Faulhaber

Eric Faulhaber wrote:

> Boris, with rev 14450, I get this in an infinite loop when I run save-cache3.p:
>
> [...]
>
> This must be what you were seeing when you reported save-cache3.p was not exiting normally. In fact, it was never getting past the SAVE CACHE statement, causing an abnormal end, and then retrying. You need to tail the server log when you run test cases. Some change in rev 14450 seems to have broken the runtime schema conversion.

On second thought, this might not be what you were seeing. This was caused in my environment because I had a table defined in my mutable database that had *only* the primary key defined, and no other columns representing legacy fields. Although we have a request to allow such a table for an edge case, we don't support that yet. Please add a validation check for this situation, so we report an error gracefully, rather than getting deeper into the runtime code and failing as posted above.

I've committed rev 14451, which uses ErrorManager to raise an error condition, rather than throwing ConditionException directly. Please use this idiom when you need to raise a 4GL-style error that gets handled by code which calls into the persistence layer.

#115 - 01/13/2023 11:56 AM - Boris Schegolev

Eric Faulhaber wrote:

> Eric Faulhaber wrote:
> On second thought, this might not be what you were seeing. This was caused in my environment because I had a table defined in my mutable database that had *only* the primary key defined, and no other columns representing legacy fields. Although we have a request to allow such a table for an edge case, we don't support that yet. Please add a validation check for this situation, so we report an error gracefully, rather than getting deeper into the runtime code and failing as posted above.
>
> I've committed rev 14451, which uses ErrorManager to raise an error condition, rather than throwing ConditionException directly. Please use this idiom when you need to raise a 4GL-style error that gets handled by code which calls into the persistence layer.

Yes, that's a different situation. But I did have a similar problem going into an infinite loop - this happens on any failure of DMO creation. I am not sure if that is a problem of the program or an issue with FWD. But it's painful to work with :D I had to clean my DB a few times to fit the state of the codebase.

BTW, other programs didn't do it, save-cache1.p and save-cache2.p did shutdown on DMO creation failures.

#116 - 01/13/2023 06:26 PM - Boris Schegolev

I pushed minor updates to the table validation procedure - new rule plus the table name shows up in the error window. Revision number is 14452.

#117 - 01/17/2023 01:28 AM - Eric Faulhaber

To determine the permissible data types, please refer to the static initializers for the fwd2sql maps in the respective dialect implementations (e.g., P2JPostgreSQLDialect.fwd2sql). This is the basis for the data type tables in the wiki. You should still follow up with the process described in #6371-104 to verify how the JDBC drivers report these types in the metadata interrogation calls, but the basic data types allowed are from those maps.

#118 - 01/17/2023 07:42 PM - Boris Schegolev

OK, I pushed proper translation of datatypes for Postgres - rev 14454.

I have to say that I don't really like the implementation, but maybe I am missing something. I feel like there's too many translations happening and all of them are based on string values. In particular, there's ParmType, ProgressParserTokenTypes, java.sql.Types, various implementations of com.goldencode.p2j.util.BaseDataType, etc. ParmType wraps BaseDataType, but the rest just has ad-hoc translations throughout the codebase (typically switch-case over string values or some map). Also, the translations are usually incomplete because they serve specific purposes like in my case, when I only need to support types used in conversions.

Should we refactor that somehow before we add implementations for other dialects?

#119 - 01/18/2023 07:39 PM - Eric Faulhaber

Boris, I agree that the messiness of the data type processing is not nice. If you have a way to simplify this while still producing a valid schema AST and ultimately DMO interface, I am happy to use it.

Higher priority, however, is this issue. Please try this recreate:

1. Define a valid test table in the mutable database, with at least 3 columns (including the surrogate primary key).
2. Launch the FWD server.
3. Run save-cache3.p and use the query for each test when prompted.
4. Dismiss the client, but do not restart the server.
5. Remove a column from the test table.
6. Run save-cache3.p again with the **same exact** query.

This should produce a PostgreSQL error (look at your server log to see the cause).

I believe it is caused by one of our levels of query caching remembering SQL generated for the query, based on the old table structure. If my reasoning is correct, we need a way to invalidate all cache entries associated with a particular table, upon detecting that the table structure has changed.

Alternatively, we could prevent all mutable database queries from being cached in the first place, but that would give up an optimization for the common use case (i.e., just using the table after its definition is settled), based on a less common use case (i.e., changing the table between executions of the same query). Depending on how much effort it looks to be to do the invalidation, we may opt for the latter approach temporarily, as we need to minimize time spent on this and get the basic, first pass implementation finished.

Other needs (some taken from your email):

1. implement MariaDB support (high in priority, but is dependent upon other work with branch 6129b)
2. execute an implicit schema check operation when a mutable database is first connected, to ensure we have the latest table definitions reflected in the loaded DMOs
3. more validations
    - ensure the database has the ID generator sequence
    - except for the implicit primary key index, the primary key may only appear at the end of a non-unique index (need to add this to the wiki)
    - others?
4. timestamp with timezone support
5. EXTENT support (currently, expanded mode only, with columns identified as being part of a legacy extent field by comment/hint)
    - this may require some clever validation
6. refactoring/cleanup (e.g., data type mess)
7. FORMAT support (very low priority, using defaults now, and 4GL code often customizes this anyway)

I think fixing the error described above and implementing the first three of these items (plus more testing, which could reveal additional issues) are the most critical to get done in order to let wider testing of this feature begin.

#120 - 01/20/2023 05:49 PM - Boris Schegolev

I found the root cause for the error you described. Luckily, it's not caching or so, but just a logical mistake (SchemaCheck uses wrong Table object when removing columns, effectively recreating exactly the same DMO). I will be able to fix it on Monday.

#121 - 01/23/2023 06:57 PM - Boris Schegolev

I pushed a fix for the issue with removing columns. Also fixed another minor thing. See revision 14456.

#122 - 01/25/2023 04:07 PM - Boris Schegolev

Implemented validation "except for the implicit primary key index, the primary key may only appear at the end of a non-unique index" - see revision 14457.

#123 - 01/26/2023 03:48 AM - Eric Faulhaber

I committed 6371a/14458, which:

- implements thread safety to prevent concurrent schema checking and DMO generation for the same database;
- implements the "mutable" configuration option in the directory, to mark a database as mutable (i.e., can be modified externally and will be scanned for changes at startup and when SAVE CACHE is executed);
- adds an implicit schema check when a mutable database is first initialized;
- cleans up the data type mappings somewhat;
- cleans up the AST registry after runtime conversion of a DMO interface;
- tidies up some formatting.

Then I:

- rebased 6371a over trunk rev 14482;
- merged 6371a into trunk and committed the merge as trunk rev. 14483;
- archived task branch 6371a.

I am working on manually merging the SAVE CACHE changes into shared branch 6129b.

To mark a database to work with SAVE CACHE, add a boolean "mutable" flag to the database's p2j container in the directory, as follows:

```
        ...
        <node class="container" name="database">
          <node class="container" name="<database_name>">
            <node class="container" name="p2j">
              ...
              <node class="boolean" name="mutable">
                <node-attribute name="value" value="true"/>
              </node>
            </node>
            ...
```

#124 - 01/26/2023 03:53 AM - Eric Faulhaber

Boris, we were discussing the datetime-tz implementation yesterday, but there is still a TODO in the code for the ID generator sequence validation. That validation is higher priority. If that sequence is not in the database, FWD will not be able to insert records into it.

#125 - 01/26/2023 06:28 AM - Eric Faulhaber

I've manually ported the SAVE CACHE feature from trunk rev 14483 to 6129b and committed that as 6129b/14383. I have done some smoke testing and it seems to work ok.

Boris, you will need to reconvert save-cache3.p and any other 4GL test cases which use SAVE CACHE. You also will need to update your directory as noted in #6371-123.

Please implement the MariaDB lenient dialect support as your top priority and make sure it works as well as the PostgreSQL dialect. If you have time, implement the sequence validation and any of the remaining features that you can today from the list in #6371-119. But first and foremost, anything you check in must not break the build (this is a shared branch) and can not regress the existing SAVE CACHE functionality. The build you commit today will likely be the build that goes into customer testing.

#126 - 01/26/2023 05:20 PM - Boris Schegolev

I pushed 2 commits into 6129b - 14385 and 14386. Now our save-cache test programs operate on PostgreSQL and MariaDB equally. I don't see any more issues.

I fixed a regression though - I had to re-introduce translateToProgressParserTokenType(ParmType) in SchemaCheck. This method returns a different set of int constants than propTypeToProgress.get(parmType), so it triggered some rules when KW_INIT (default value) was added to the DMO. This is one of those cases we discussed with Eric, when the mapping is clearly an overhead, but there's more logic hidden in those mappings.

#127 - 01/27/2023 07:21 AM - Eric Faulhaber

Boris Schegolev wrote:

> I pushed 2 commits into 6129b - 14385 and 14386. Now our save-cache test programs operate on PostgreSQL and MariaDB equally. I don't see any more issues.

I tested this on a cluster (is that the right word for Maria?) with multiple databases on it, including my mutable test database, which did not yet have any tables, indices, or sequences in it.

Unfortunately, the schema check is not limited to only the database for which we make the connection. ALL the tables in ALL the databases were scanned (over 1300). We need a way to filter only for the target schema represented by the database name passed into SchemaCheck.run().

I also made these adjustments:

- allow for no DF file to be present for a mutable schema during conversion;
- allow the data types we convert to in MariaDbLenientDialect.

See 6129b/14388.

#128 - 01/27/2023 10:17 AM - Boris Schegolev

Filtering is resolved - see revision 6129b/14389. It also removes the TODO I had there.

#129 - 01/27/2023 10:44 AM - Boris Schegolev

I added same filtering for columns and indexes, it should save some CPU time. Also, MariaDB returns boolean as bit, so I put it back on the list of datatypes.

Rev number is 6129b/14390.

#130 - 01/27/2023 11:51 PM - Eric Faulhaber

Rev 6129b/14392 fixes issues with setting initial values in the DMO interface. It also pulls in a newer MariaDB JDBC driver (3.1.2 GA).

#131 - 01/30/2023 09:42 AM - Greg Shah

The documentation for this task is in Non-Standard Save Cache Implementation.

#132 - 01/31/2023 02:13 AM - Eric Faulhaber

Boris, please continue implementing the remaining items in #6371-119.

I've also noticed in the code that the column comment (if any) returned by JDBC metadata is being used as the legacy attributes LABEL and COLUMN-LABEL. Considering that we are using the comments to embed hints, we don't want to use these comments in their entirety for these attributes. Please allow these attributes to be specified via hints, like the data type overrides are.

#133 - 01/31/2023 03:55 PM - Boris Schegolev

I pushed revision 6129b/14395 with comment cleaning procedure and some JavaDoc that was missing.

#134 - 02/01/2023 06:11 PM - Boris Schegolev

Branch 6129b is frozen for now, so, let's take a look at what we know was not implemented by now:

- more validations - ensure the database has the ID generator sequence - **IN PROGRESS**, but I'm not sure how we want to approach this in other dialects. My guess is that we use auto_increment property in MySQL/MariaDB PKs. It is not a sequence, so what do we validate here? Do we just skip the validation if dialect does not support sequences?
- timestamp with timezone support - **IN PROGRESS**
- EXTENT support (currently, expanded mode only, with columns identified as being part of a legacy extent field by comment/hint)
  this may require some clever validation
- FORMAT support (very low priority, using defaults now, and 4GL code often customizes this anyway)

#135 - 02/02/2023 08:27 AM - Greg Shah

> EXTENT support (currently, expanded mode only...

I think we only need expanded mode support. The old "denormalized" approach is going away. Let's not waste time implementing it here.

#136 - 02/02/2023 01:18 PM - Eric Faulhaber

Boris Schegolev wrote:

> Branch 6129b is frozen for now, so, let's take a look at what we know was not implemented by now:
>
> - more validations - ensure the database has the ID generator sequence - **IN PROGRESS**, but I'm not sure how we want to approach this in other dialects. My guess is that we use auto_increment property in MySQL/MariaDB PKs. It is not a sequence, so what do we validate here? Do we just skip the validation if dialect does not support sequences?

We do not use auto_increment, because we have to create a primary key before we create the record in the database. There is internal work we do with the key before we ever save the record. We use the p2j_id_generator_sequence to generate new primary keys; it must be present in the database, as documented in the SAVE CACHE wiki. So far, all dialects we support implement sequences.

- FORMAT support (very low priority, using defaults now, and 4GL code often customizes this anyway)

It seems like this can be supported through the comment mechanism, like any other legacy attribute we cannot glean from JDBC metadata.

#137 - 02/06/2023 02:50 PM - Boris Schegolev

I am working on support for sequences in SchemaCheck for use in validations and I wanted to have a confirmation that I have the right direction.

The situation is that sequences are not generally supported by JDBC API. I saw a simple trick that I initially intended to use, but it's Oracle specific and doesn't work with latest drivers. Also, MySQL doesn't support sequences and MariaDB only added them in 10.3 (for example, my Linux box installed 10.1 by default).

Currently, my best option is adding SQLs to retrieve sequence information into individual dialects like we did with data types. Would it be OK like that or do you see better options?

#138 - 02/06/2023 03:50 PM - Eric Faulhaber

Boris Schegolev wrote:

> Currently, my best option is adding SQLs to retrieve sequence information into individual dialects like we did with data types. Would it be OK like that or do you see better options?

This would be ok; I don't see a better way. Note that this isn't a hard requirement to make the SAVE CACHE implementation work. However, it is a useful validation, so that a user is aware early that there will be a problem later (i.e., when using converted application code to create new records for

the mutable database), unless they create the appropriate sequence.

#139 - 02/13/2023 06:17 PM - Boris Schegolev

- File 6129b-sequences.diff added

Attaching a working version of sequences support in SchemaCheck and SchemaValidator for review.

#140 - 02/21/2023 03:40 PM - Eric Faulhaber

Please create a 6371b branch from trunk for this patch and the other remaining features. Thanks.

#141 - 02/27/2023 05:05 PM - Boris Schegolev

Tested and pushed Sequences and validations as revision 6371b/14485. Proceeding with timezone support.

#142 - 02/28/2023 05:33 PM - Boris Schegolev

Pushed initial implementation of DTTZ support in SchemaCheck as revision 6371b/14486. I am just not sure where the value (timezone offset) belongs - there doesn't seem to be an annotation type for it. But maybe I'm just missing something.

#144 - 03/24/2023 06:56 PM - Boris Schegolev

I puhsed revision 6371b/14487. It provides support for DTTZ columns, fixes for MariaDB in SchemaCheck and support for EXTENT in both normalized and denormalized form. I'll do more testing for corner cases, but it seems to do what it's supposed to. Also, extent fields now rely on naming convention only, let's see how it behaves. More rules or annotations can be added to make it more robust if needed.

#145 - 03/28/2023 07:05 PM - Boris Schegolev

I added support for comments / meta info for extent fields as revision 6371b/14488. Documentation is also updated - see https://proj.goldencode.com/projects/p2j/wiki/Non-Standard_Save_Cache_Implementation#section-7

#146 - 03/29/2023 03:30 PM - Eric Faulhaber

Code review 6371b/14485-14488:

Overall, it seems good functionally; nice work. I do have some considerations and comments.

Extent fields:

> ...and support for EXTENT in both normalized and denormalized form

As noted in #6371-100, #6371-101, and #6371-135, we are not intending to support the "normalized" extent field approach in mutable databases (or at all, anymore). I don't want to complicate the implementation with this, nor have to maintain this support going forward. If a table with this name pattern is scanned, please log a warning that such a table appears to represent a deprecated extent field implementation. Go ahead and process the table anyway *as a regular table*, in case it is just a coincidentally, oddly named table. If an error occurs, this warning gives an indication of why things went wrong.

> ...extent fields now rely on naming convention only, let's see how it behaves

This is a little concerning, since we don't control the column naming in a mutable database, and someone creating a schema easily could name a series of non-extent fields with an underscore and sequential numeric values. It looks like your heuristic looks for 3 such fields as a threshold to considering these extent field elements. How much effort is it to implement a comment requirement to define an extent field?

Testing:

- How much testing have you done?
- How have you tested the datetime-tz, extent field, and sequence changes?
- You mention corner cases. Have you tested cases of updating an existing extent field or sequence (as opposed to adding new or deleting existing instances)?

- Have you implemented 4GL test cases using SAVE CACHE, or just Java unit tests?

Rebase:

6371b needs a rebase; trunk is now at rev 14519, while this branch is based on 14484. I think That's a lot of change; hopefully, a rebase does not break anything. Please re-run your tests afterward.

Other:

All updated files need header entries. Since this branch is based on a new trunk revision and will be represented by a single trunk merge/commit, you would add a number to these entries (we didn't previously, because we were making many changes on top of shared branch 3821c). If the copyright notice still is for 2022, please change it to 2023, or if the file's life spans multiple years, the end year should be 2023 (e.g., ComparisonResult should be changed from "2022" to "2022-2023").

All fields, methods, etc. (even private) should have javadoc. We missed some in earlier commits. Please clean these up.

Please honor the coding standards w.r.t.:

- the line length limit of 110 characters;
- the order of fields in a class (i.e., static before instance, and within that, ordered by accessibility modifiers from most accessible to least accessible).

For future reference: we generally avoid streaming over collections in code that may be performance sensitive. There is a lot of this idiom in the SAVE CACHE implementation. I don't think this is a particularly performance sensitive area of FWD, so we should be ok.

#147 - 03/30/2023 01:17 PM - Boris Schegolev

I rebased the branch (well, merged trunk into 6371b) without major issues.

> As noted in #6371-100, #6371-101, and #6371-135, we are not intending to support the "normalized" extent field approach in mutable databases (or at all, anymore). I don't want to complicate the implementation with this, nor have to maintain this support going forward. If a table with this name pattern is scanned, please log a warning that such a table appears to represent a deprecated extent field implementation. Go ahead and process the table anyway *as a regular table*, in case it is just a coincidentally, oddly named table. If an error occurs, this warning gives an indication of why things went wrong.

Firstly, the normalized support was easier to implement. Secondly, I can set it to raise an error instead of pulling the table in, although I suspect there could be some customer who already has a database with a normalized extent table. They may want to make this DB mutable, but the functionality will be broken until they convert to denormalized mode.

Anyway, it's up to you. I don't have a strong feeling about it. I can just remove the method and raise an error instead. Or keep the method and still show some warning.

> It looks like your heuristic looks for 3 such fields as a threshold to considering these extent field elements. How much effort is it to implement a comment requirement to define an extent field?

Comment/Meta support is already implemented. We can just set the threshold higher (let's say 9999) and FWD will require any extent field to have a

comment/meta information to work. The comment structure itself is documented here:
https://proj.goldencode.com/projects/p2j/wiki/Non-Standard_Save_Cache_Implementation#section-7

- How much testing have you done?
- How have you tested the datetime-tz, extent field, and sequence changes?

I did manual testing on all of those features. Adding and removing columns, rules for sequences, etc.

- You mention corner cases. Have you tested cases of updating an existing extent field or sequence (as opposed to adding new or deleting existing instances)?

This is still in progress. I need to check changes in extent field (i.e. changed data type) to see how it's reflected in difference list. With high probability, it will work, I just didn't check that explicitly.

- Have you implemented 4GL test cases using SAVE CACHE, or just Java unit tests?

I have a simple 4GL case of my own and the original save-cache-3.p 4GL test case. I didn't add any new unit tests recently (or run them for that matter).

> All updated files need header entries. Since this branch is based on a new trunk revision and will be represented by a single trunk merge/commit, you would add a number to these entries (we didn't previously, because we were making many changes on top of shared branch 3821c). If the copyright notice still is for 2022, please change it to 2023, or if the file's life spans multiple years, the end year should be 2023 (e.g., ComparisonResult should be changed from "2022" to "2022-2023").

Done.

> All fields, methods, etc. (even private) should have javadoc. We missed some in earlier commits. Please clean these up.
>
> Please honor the coding standards w.r.t.:
>
> - the line length limit of 110 characters;
> - the order of fields in a class (i.e., static before instance, and within that, ordered by accessibility modifiers from most accessible to least accessible).

Yes, I'll push additional cleanup after testing.

#148 - 03/30/2023 01:37 PM - Eric Faulhaber

Boris Schegolev wrote:

> I rebased the branch (well, merged trunk into 6371b) without major issues.

This is not the same as a rebase. After a rebase, the task branch should have all the individual trunk commits before the first task branch commit. I expect the merge of trunk into the task branch will now cause problems and lose information when we try to merge the task branch back into trunk. Are you able to revert to 6371b/14488 and do a true rebase?

#149 - 03/30/2023 02:42 PM - Eric Faulhaber

Eric Faulhaber wrote:

> Boris Schegolev wrote:
>
> > I rebased the branch (well, merged trunk into 6371b) without major issues.
>
> This is not the same as a rebase. After a rebase, the task branch should have all the individual trunk commits before the first task branch commit. I expect the merge of trunk into the task branch will now cause problems and lose information when we try to merge the task branch back into trunk. Are you able to revert to 6371b/14488 and do a true rebase?

Boris, as discussed over email, I overwrote 6371b with rev 14488 from yesterday, then I rebased over trunk rev 14523. 6371b is now at rev 14527.

#150 - 03/30/2023 02:44 PM - Boris Schegolev

Awesome, I will apply headers update and continue cleanup and testing. Thank you!

#151 - 03/30/2023 02:51 PM - Eric Faulhaber

Boris Schegolev wrote:

> I rebased the branch (well, merged trunk into 6371b) without major issues.

As noted in #6371-100, #6371-101, and #6371-135, we are not intending to support the "normalized" extent field approach in mutable databases (or at all, anymore). I don't want to complicate the implementation with this, nor have to maintain this support going forward. If a table with this name pattern is scanned, please log a warning that such a table appears to represent a deprecated extent field implementation. Go ahead and process the table anyway *as a regular table*, in case it is just a coincidentally, oddly named table. If an error occurs, this warning gives an indication of why things went wrong.

> Firstly, the normalized support was easier to implement. Secondly, I can set it to raise an error instead of pulling the table in, although I suspect there could be some customer who already has a database with a normalized extent table. They may want to make this DB mutable, but the functionality will be broken until they convert to denormalized mode.

Currently, all applications relying on this SAVE CACHE implementation also are using expanded (i.e., "denormalized") extent mode. We are endeavoring to get all projects on that mode.

> Anyway, it's up to you. I don't have a strong feeling about it. I can just remove the method and raise an error instead. Or keep the method and still show some warning.

Please show the warning if the table name meets the "normalized" secondary table naming pattern, but attempt to process the table as a regular table.

> It looks like your heuristic looks for 3 such fields as a threshold to considering these extent field elements. How much effort is it to implement a comment requirement to define an extent field?

> Comment/Meta support is already implemented. We can just set the threshold higher (let's say 9999) and FWD will require any extent field to have a comment/meta information to work. The comment structure itself is documented here:
> https://proj.goldencode.com/projects/p2j/wiki/Non-Standard_Save_Cache_Implementation#section-7

Good idea. Please do that.

- How much testing have you done?
- How have you tested the datetime-tz, extent field, and sequence changes?

> I did manual testing on all of those features. Adding and removing columns, rules for sequences, etc.

- You mention corner cases. Have you tested cases of updating an existing extent field or sequence (as opposed to adding new or deleting existing instances)?

> This is still in progress. I need to check changes in extent field (i.e. changed data type) to see how it's reflected in difference list. With high probability, it will work, I just didn't check that explicitly.

- Have you implemented 4GL test cases using SAVE CACHE, or just Java unit tests?

> I have a simple 4GL case of my own and the original save-cache-3.p 4GL test case. I didn't add any new unit tests recently (or run them for that matter).

I'm not sure the save-cache-3.p will work with extent fields (at least, I didn't write it with that in mind). I guess it should work with datetime-tz fields. The sequence changes are just validation, so I would expect it to continue to work with the sequence check. If your new test case covers extent fields, we should be ok.

#152 - 03/31/2023 12:08 PM - Constantin Asofiei

I've looked at 6371b and the changes look safe, they are limited to SAVE CACHE package in FWD.

#153 - 03/31/2023 01:53 PM - Boris Schegolev

I pushed cleanup and headers update for modified files as revision 14529. Also, normalized extent support is replaced with a warning and denormalized extent requires comments/meta information. We should be good to go.

#154 - 03/31/2023 02:18 PM - Eric Faulhaber

- % Done changed from 80 to 100

- Status changed from WIP to Review

Code review 6371b/14529: the changes look good.

#155 - 03/31/2023 03:06 PM - Boris Schegolev

Merged into trunk as revision 14524.

#156 - 04/24/2023 06:50 AM - Jurjen Dijkstra

I was asked to test this statement. I will not be able to have a look at this soon, probably not even before summer vacation.
Beside that, I dont work with postgress or java, so I don't understand why I was asked in the first place, you'd much better ask someone else.

#157 - 04/24/2023 05:30 PM - Eric Faulhaber

Code review 6371d/14544:

One **very minor** change: please fix the open curly brace at SchemaComparator.java:149 to comport with coding standards. Other than that, this can be merged to trunk (presumably after another rebase).

#158 - 06/12/2023 09:41 AM - Greg Shah

- Status changed from Review to Test

#159 - 09/06/2023 07:00 AM - Greg Shah

- Status changed from Test to Closed

Files

6371.diff	5.4 KB	07/14/2022	Boris Schegolev
boris.p	921 Bytes	07/27/2022	Boris Schegolev
SchemaComparator.java	4.25 KB	08/01/2022	Boris Schegolev
SchemaComparator.java	7.98 KB	08/05/2022	Boris Schegolev
save-cache1.p	215 Bytes	08/26/2022	Eric Faulhaber
save-cache2.p	53 Bytes	09/29/2022	Eric Faulhaber
6129b-sequences.diff	10.7 KB	02/13/2023	Boris Schegolev